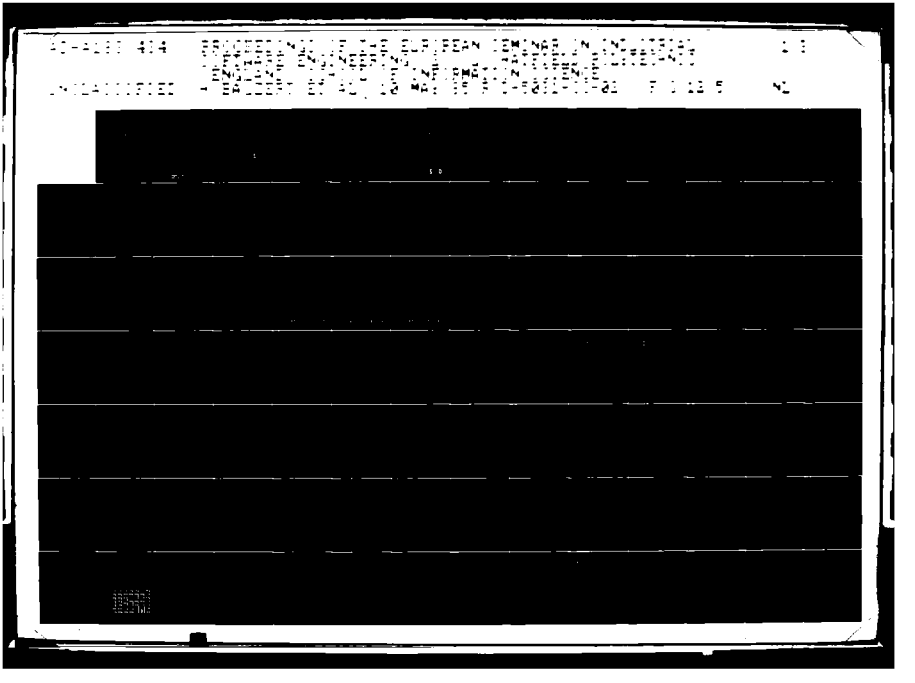
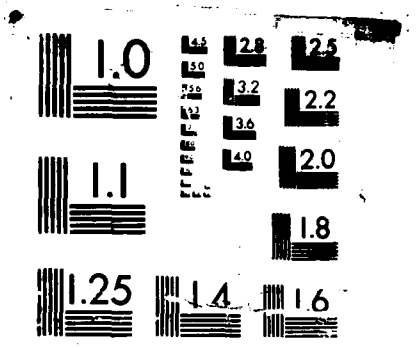




MICROCOPY RESOLUTION TEST CHART

PROCEEDINGS OF THE

SECOND EUROPEAN SEMINAR ON INDUSTRIAL SOFTWARE ENGINEERING

Freiburg, West Germany

9 - 10 May 1985

COMPONENT PART NOTICE

THIS PAPER IS A COMPONENT PART OF THE FOLLOWING COMPILATION REPORT:

TITLE: Proceedings of the European Seminar on Industrial Software Engineering (2nd) Held in Freiburg (Germany, F.R.) on 9-10 May 1985.

TO ORDER THE COMPLETE COMPILATION REPORT, USE AD-A183 434.

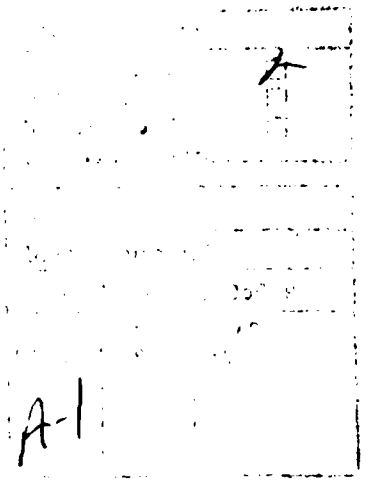
THE COMPONENT PART IS PROVIDED HERE TO ALLOW USERS ACCESS TO INDIVIDUALLY AUTHORED SECTIONS OF PROCEEDING, ANNALS, SYMPOSIA, ETC. HOWEVER, THE COMPONENT SHOULD BE CONSIDERED WITHIN THE CONTEXT OF THE OVERALL COMPILATION REPORT AND NOT AS A STAND-ALONE TECHNICAL REPORT.

THE FOLLOWING COMPONENT PART NUMBERS COMPRISE THE COMPILATION REPORT:

AD#: P005 554 Thru AD#: P005 566

AD#: _____ AD#: _____

AD#: _____ AD#: _____



DTIC FORM 463 MAR 85

OPI: DTIC-TID

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS AD-A183434
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) R&D 5031-CC-02
6a. NAME OF PERFORMING ORGANIZATION The Hatfield School of Information Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION USARDSG(UK)
6c. ADDRESS (City, State, and ZIP Code) PO Box 109 Hatfield Herts AL10 9AB	7b. ADDRESS (City, State, and ZIP Code) Box 65 FPO NY 09510-1500
8a. NAME OF FUNDING/SPONSORING ORGANIZATION USARDSG(UK) ARO-E 8b. OFFICE SYMBOL (If applicable) AMXSN-UK-ZA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAJA45-85-M-0219
8c. ADDRESS (City, State, and ZIP Code) Box 65 FPO NY 09510-1500	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 61102A PROJECT NO.: 1L161102BH57 TASK NO.: 03 WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
(U) Proceedings of the Second European Seminar on Industrial Software Engineering

12. PERSONAL AUTHOR(S) H. Balzert, M. I. Jackson, P. Dencker, H.S. Jansohn, G. Goos, E. J. Neuhold D. G. Morgan, H. Weber, R. Popescu-Zeletin, G. Le Lann, M. R. Moulding, J. Favaro, J. Hall

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Proceedings	FROM 9 May 85 TO 10 May 85		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
09	02		(U) Software Engineering

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

It is now common knowledge that the announcement of the Japanese Fifth Generation Computing Programme in 1982 led to the announcement of a number of national and international programmes in Information Technology, including the Esprit programme. These are popularly considered as being in response to the Japanese initiative. However, the Esprit and the UK Alvey programmes had been preceded by some years of discussion between industrial and government representatives. They, therefore, overlap rather than follow the Japanese Programme and tend to cover a much wider range of subjects. The Esprit Software Technology Programme 1985 is described, followed by a report on the Japanese Fifth Generation Conference. A comparison between the approaches of the two programmes is then made.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☒ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. James W. Gault	22b. TELEPHONE (Include Area Code) 01-409 4423 22c. OFFICE SYMBOL AMXSN-UK-RI

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

INDUSTRIAL SOFTWARE TECHNOLOGY '85

CONTENTS

(*Unix is a trademark of AT&T Bell Laboratories)

THE ESPRIT SOFTWARE TECHNOLOGY PROGRAMME 1985

and THE JAPANESE FIFTH GENERATION CONFERENCE 1984

CONTRASTING APPROACHES TO RESEARCH INTO INFORMATION TECHNOLOGY

D.G. MORGAN Research Director (Software)
Plessey Electronic Systems Research Ltd., Roke Manor.

The views set out in this paper are my personal opinions and should not be taken as representing an official view held either by the Plessey Company p.l.c., or by the C.E.C.

It is now common knowledge that the announcement of the Japanese Fifth Generation Computing Programme in 1982 led to the announcement of a number of national and international programmes in Information Technology, including the Esprit programme. These are popularly considered as being in response to the Japanese initiative. However, the Esprit and the UK Alvey programmes had been preceded by some years of discussion between industrial and government representatives. They, therfore, overlap rather than follow the Japanese Programme and tend to cover a much wider range of subjects. Other differences in motivation, funding and industry participation mean that comparisions between the programmes are similar, to improve the national competitiveness in Information Technology, such comparisions must be made.

This paper started out as two separate papers but, as the planners of the EWICS Conference placed them one after the other in the programme, it seemed sensible to merge the two papers. Having brought them together, an obvious next step was to compare the two approaches. The form of the paper reflects this history. The Esprit Software Technology Programme 1985 is described, followed by a report on the Japanese Fifth Generation Conference. A comparision between the approaches of the two programmes is then made.

ESPRIT SOFTWARE TECHNOLOGY PROGRAMME 1985

What follows describes the work which was undertaken by the Software Technology Advisory Panel in changing the shape of the Esprit Software Technology Programme in preparation for bids to be received during 1985.

Why should there be a need to restructure the Software Technology Programme for 1985? The Esprit Software Technology Programme had been in place for two years and a considerable reponse had been received both to pilot projects and to the first call for proposals. However, despite substantial numbers of applications, the overall response to the first call for proposals was seen as disappointing both by the Technical Panel and by the Commission. Disappointing both in quality and quantity.

When the Esprit project was originally conceived, Software Technology was seen as being one of the major enabling technologies required by the Information Technology business. On all sides people complain about the lack of productivity in software development and the lack of availability of skilled staff. Consequently when the programme was formulated it was allocated funds similar to those allocated to the other major areas in the overall Esprit Programme. When the call for proposals went out it was made clear that the Commission expected an enthusiastic response.

In formulating the shape of the Esprit programme the Commission had taken the advice of an advisory panel which contained representatives of many of Europe's leading electronics companies that 75% of the projects should be large scale projects; that is of greater than ten million ECUs. It was pointed out strongly by the Software Technology Panel of the day that this was inappropriate to the state-of-the-art of Software Technology and the sort of projects that were judged to be required in order to advance this field. However the Commission's reply was that the proportion of large scale projects applied to the Esprit Programme overall and was not expected necessarily to apply to each sub-programme. In principal therefore, this would leave scope for a number of smaller projects in Software Technology. In the event the evaluation team the Commission established rejected more than 50% of the proposals received in the field of Software Technology. Sixteen projects were placed, four projects were, after reconsideration, reassessed as being suitable for support providing that their scope was reduced and these projects were asked to re-submit during the last year. In consequence the budget allocated to Software Technology was not utilised.

It is worth examining why there was this rejection of such a large number of projects when the Esprit programme had received such major publicity and why Software Technology in particular has such a poor response. There are many explanations and the following list is a personal selection of what I think are the main problems:

A. It was felt that the Assessment Panel had applied rather too academic a standard to the evaluation of the proposals. This was a view I had held myself at the time. However, the Commission did conduct an enquiry subsequently into the performance of the Assessment Panel and satisfied itself that it had in fact exercised its remit fairly.

B. It has become clear that many smaller companies did not feel that they were able to put in the investment needed in order to enter a successful Esprit bid, and were disccuraged by the thought of mounting a ten million ECU project of which they and their partners had to find 50% funding. There were smaller companies prepared to join consortia who were unable to find partners.

C. Many proposals that were submitted were badly written. They failed to meet the most elementary requirements of a proposal in not having clearly identified objectives, timetables or evaluation criteria.

D. The programme had been structured into a number of different areas and bidders were expected to indicate the area into which their proposal fitted. It was a comment of the evaluation team that many proposals did not seem to fit clearly into one area or another and they attributed this to the non-specific nature of the published Software Technology programme in the call for proposals in 1984. It is worth commenting here that this lack of specificity had been a deliberate policy of the Software Technology Panel. It had been felt that, in such a rapidly growing field, to impose too rigid a view as to what research programmes should be undertaken was to curtail the inventivness of proposals. In the event this was probably a mistake, although at the time seemed to be a very sensible approach. In drawing up the new 1985 Programme this was particularly addressed.

E. Industry was not prepared to invest large sums of money in collaborative, pre-competitive research into Software Technology, either because they were not investing in Software Technology at all, or because it was felt to be too competitive a subject.

As a consequence of this poor response, in contrast to some very enthusiastic reponses in other areas such as computer integrated manufacture, the Commission came under pressure from industrial representatives to consider the reallocation of money within the programme. The argument being that clearly there was not the interest in developing the field of software technology. It is to the Commission's credit that they resisted such movements in the last year and in fact encouraged the technical advisory panel to create a new programme which gave the opportunity to bidders to catch up with the lack of support they had given in the first year.

When the Technical Panel met again to start the consideration of the programme for 1985, the facts given above were presented to them and they were asked to reconsider what shape the programme should have in order to achieve a more satisfactory response in 1985. In discussions which followed at Esprit Technical Week and other occasions, it became very clear that there were two main criticisms of the exisiting Software Technology Programme.

1. The published programme was thought to be too vague and was too sub-divided and therefore appeared more complex than was intended. It was difficult for proposers to focus on one aspects of the programme.

2. Rather than needing to push the frontiers of Software Technology forward, the real problems facing Software Technology were that there was insufficient practice and use of existing methods and tools let alone the need to develop more advanced tools. This had not been identified in the published programme.

It is clear that on the surface this is not a particularly glamorous message to put into what is thought to be a very forward looking research programme. However, it is a commonly voiced slogan that Software Technology is about creating a new discipline of software engineering and engineering is about practical skills. Examination of the world literature in the applications of software engineering find relatively few papers on the comparative evaluation of one technique against another in terms of the practical development of large programmes. Much of the reason for this is, of course, the great difficulty in carrying out effective evaluations. In redrafting the Software Technology Programme therefore these criticisms needed to be countered. A very large software project contains all the problems of any project which requires the co-ordination of the efforts of some hundreds of human beings with widely different talents and personality; all of whom are attempting to carry out an extremely difficult and complex task to what are usually very tight time scales. Very few

projects are sufficiently alike to allow for close comparision of different methods between different projects. Much of the current drive of Software Technology is mainly an act of faith that the techniques that are being introduced will lead to very much more efficient software production. Certain trends, such as that towards formality, appears self evidently beneficial as leading to greater rigour and hence to higher quality and reduced testing time on software.

Much of the published work over the last decade has indicated that any large software projects have the majority of its money spent during the period following the implimentation of the first version of the software. However, it is perhaps not commonly appreciated how government and commercial policy can affect the shape of that curve. I am told by an emminent U.S. Government official that many of the projects on which the early work of evaluating life cycle costs were based, were in fact projects created in an environment which allowed contractors to develop complex unique software using their own standards, and in many cases their own tools. The customer (in most cases the US Government) would then be required to turn to that organisation in order to have the maintenance and post design services carried out. Commercial pressures and the need to keep a maintenance team on software that was rapidly going out of date because of the very rapid developments of hardware technology, meant that in many cases the costs of such maintenance rose very rapidly. It is thought that this could well have dictated the shape of some those early urves.

It is interesting to compare the current attitudes in many countries towards computer-aided circuit design with that towards software tools. There seems to be little question as to the desirability of software tools to aid complex design, but little evidence that software tools are considered as useful. The ability to measure, quantitatively, the product in the former case must be a major factor. Until more effective metrics are available, the introduction of software technology will be hampered by the inability to be justified in commercial terms.

To turn once again to the 1985 programme, after recognising these omissions of the previous programmes it was decided that a new strategy should be adopted in formulating the programme for 1985. This strategy can be summarised as follows:

1. Software Technology was agreed to be still a major enabling technology of systems design. Its main purpose will be to assist in the more rapid introduction of new products and in the reduction in total lifecycle costs.

2. Taken as a whole, Europe had an expertise in software technology that was the equal, if not superior to that in Japan and the United States. However, its problem was that the expertise was spread widely throughout Europe and the aim of the Esprit programme should be to co-ordinate this expertise and to ensure that it was adopted by industry.

3. The reasons for the lack of adoption by industry of existing techniques were examined and it was felt that there were two major factors, one was the cost of implementation of many of the modern techniques and secondly, the lack of information as to the benefits to accrue from these techniques available to middle management. That is, the Project Managers within organisations who, faced with the need to set up a project team for a new project will, in general, use those techniques with which they are familiar and whose benefits have already been proven. The lack of such information at that level would obviously be a major barrier to the adoption of new techniques.

4. Consequently, it was decided that a new class of projects should be brought in to introduce software technology and to evaluate and disemminate the effectiveness of the technology.

A programme was therefore re-cast:

1. To provide a more precise statement of the desired projects.

2. To fit in with projects already placed including pilot projects

3. To alter the balance of projects towards supporting the adoption of software engineering by industry.

This gave rise to a programme of the following shape:

The old matrix structure was thought to be too complex and was simplified so that three areas were retained and a fourth introduced:

1. Theories, methods, tools.

2. Management and industrial aspects.

3. Common environment.

4. The concept of demonstrator projects which would allow for the benefits of this technology to be demonstrated.

In reviewing this programme it was felt that the basic development of Support Environments and Tools were well covered by the SPMMS Project and the PCTE Project. However, it was felt that new work was required in:

1. Integration of hardware and software design.

2. Alternative and complementary methods of software development (in particular the co-ordination of the artificial intelligence work which was tending to use functional and logical languages).

3. Software engineering for small highly critical software (it was felt that although most of the attention was being devoted towards large scale software production there were equally critical areas where a piece of software, not in itself very large, would however form a very critical part of a piece of equipment - for example an ignition system for a motor car).

4. Metrics for software and for methodologies (this was intended to be part of the greater examination of the evaluation of software problems).

5. The man-machine interface problem for tools and environment (this was seen as being an attack on the major problems of the cost of the introduction of many of these techniques).

6. Projects covering a wider application area.

7. The evaluation projects referred to above.

The shape of this programme was welcomed by the Commission as being a more hard headed approach to the problems of software engineering and the programme was given initial acceptance. However, a number of members of the Commission and representatives of major electronics companies then visited the Japanese Fifth Conference in November and what they saw there, reinforced the Commission's need to re-examine the aims of the overall Esprit Programme when they returned to identify more demonstration projects.

They were very impressed with the clear cut goals of the Japanese Fifth Generation Programme which had been set at the outset and which was clear, had been adhered to. This contrasted with the very wide ranging aspirations of the Esprit Programme which covered a very much wider range of activities than that of the Japanese. It was felt that the Esprit Software Engineering and the AIP Programmes would benefit from an even more focused approach in the future and so efforts were made in the first few months of this year to draw up a further refinement of the aims of the Software Technology Programme and, in particular encourage industry to get together in larger groupings to form large

scale demonstrator projects. In the preliminarly discussions however, it became clear that many of the large companies who had been enthusiasic supporters of the programme in its early years were now seeing themselves as being faced with a resource shortage when faced with the need to provide more support for new, very large co-operative Software Technology Programmes, in particular in the absence of the benefits referred to above. On the other hand, nobody seemed to have found the secret of encouraging smaller comapnies to join, although various ideas had been suggested - for example - that a large industrial organisation should act as the focal point for a number of smaller proposals which would be gathered together under the managment of the larger organisation.

At the same time attempts were madde to identify quantitative goals for the Software Technology Programme. The Japanese had announced at the Fifth Generation Conference that they were considering starting a software technology programme with the declared aim of changing the degree of automation in the process of software development from 8% to 80%. Attempts were made, in a series of special meetings of technical representatives of major electronics and telecommunications companies, to draw up a similar set of goals for the Esprit Programme. Some reluctance was shown by representatives based upon:

a. The lack of true measurements of current techniques against which to compare improvements.

b. The usefullness of such simplified criteria in the current state of software development.

c. the difficulty of adding such criteria to a programme now well advanced, which had not been started with these criteria in mind.

However, some targets were suggested although these have not yet been officially published.

Further, it was considered that, in the major projects PCTE, SPMMS, GRASPIN, together with a new project to build tools backed upon PCTE, the Esprit project had a major initiative in the mainstream of modern thinking on the future of software technology.

While there is much to be said for this line now being taken by the Commission, it must be said that it was really rather too late to alter the overall direction of the Software Technology Programme for 1985. Only, when the results of this year's bids are revealed will we be able to see whether in fact this initiative to encourage industrial grouping to form major strategic projects has been sucessful.

.....

David G. Morgan
30/04/85

THE JAPANESE FIFTH GENERATION CONFERENCE IN 1984

What follows is a personal impression of the Japanese Fifth Generation Conference amplified by impressions of a number of visits made to Electronic Industry Research Labs during the previous week.

History will show, I suspect, that the Japanese Fifth Generation initiative has had a very major effect upon the interest in Information Technology in the western world and possibly, in the world at large. This interest was shown by the hundred per cent oversubscription to the Conference which was held in November of last year in Tokyo. The Japanese openness in publicising the aims of their project led to great eagerness amongst delegates to see just how far the Japanese have progressed along their chosen path. However, in the two years that the programme has been running, the world's press have, it would appear, managed to embellish the aspirations of the Japanese project with the impression that the Japanese were making a determined attempt upon the Artificial Intelligence problem. I was not present at the first conference on the Fifth Generation project but I am assured by those who were, that the impression was given at that time that the Japanese certainly intended to produce "Thinking Computers". It was very noticeable however in the opening speeches of the Conference both by Dr. Fuchi and Prof. Moko-oka that they were eager to dispell any ideas that the Japanese had attempted in the last two years to make any attack in this direction. In fact Dr. Fuchi went so far as to say that they were not tackling the Artificial Intelligence problem but they were preparing themselves to generate hardware and software which would be the next generation of the way in which Information Technology was implemented.

As the conference progressed many official speakers stood up and repeated that theme and stressed that the attack on the Artificial Intelligence problem would come as a result of international co-operation and that there remained many years work investigating the application areas of the Fifth Generation Technology that the Japanese were developing before anything approaching Artificial Intelligence would be seen.

What then have the Japanese achieved in the two full years that the project has been underway? First of all one has the general impression that they have apparently achieved all their hardware targets for the first stage and I will be talking about those a little later on. They appear to be particularly strong in hardware design and in the operating systems software and they have developed a number of products which are of a commercial standard and of wide applicability in their own right.

It was very noticable that they have a new generation of engineers widely read in the current literature. Comments were made by more than one US researcher that, the US AI community would be hard-pushed to match the number of young post-graduates who were presenting papers during the Conference. It is now probably well understood that the Fifth Generation Programme is run by a central organisation known as ICOT to which companies and state research labs have contributed staff who work together under a director Dr. Fuchi, towards achieving the goals of the Fifth Generation Programme. What is perhaps not quite so widely understood is that nearly every company that contributes staff to the central project, has also got in-house research programmes which parallel Fifth Generation Projects. Each company's programme may not be of such wide ranging applicability as the Fifth Generation Project, but in general will be a sub-set of those activities which that particular company feels is relevant to its commercial future. Since the major Japanese companies are intensely competitive, it is quite likely that there are three or four identical developments going on within Japanese industry. These are not just replications of the ICOT programme, but represents an extension and exploitation of the ICOT programme and builds upon the experience being obtained by the engineers contributed to that central team. So, for example, while the personal sequential inference machine being produced by Mitsubishi for ICOT has a performance of 100K Lips Mitsubishi are producing a similar machine with voice response, image understanding and possibly faster performance. At the same time, independently of ICOT, a further programme is being undertaken by the Nippon Telegraph and Telephone Company, the state owned PTT which has a programme underway which is probably larger than the Fifth Generation itself.

The total picture therefore is of a well focused project which is achieving its goals but because of the infrastructure of industry and research within Japan is also creating a very wide and deep understanding of the problems of developing these sort of systems and is providing a very large industrial base on which any future developments can be placed. It is not at all clear that similar strength and depth is being created either in Europe or in the United States. Overall therefore one can say that the conference was an impressive statement of the efficiency of the Japanese industrial machine.

What are the specific achievements? Figure 1 - an oft quoted diagram, shows the way in which they intend to develop their system. They are building a hardware base consisting of a relational database mechanism, an inference mechanism, sophisticated interface hardware, all of which linked together in an inference machine, of which there are going to be two types, firstly a sequential inference machine and eventually a parallel inference machine. Much has been said in the press about use of Prolog, but it was very carefully stressed that Prolog has been adopted as the operating system language for their hardware and that they have not committed themselves to using Prolog alone for the implementation of knowledge based systems, in fact many of the technical papers given at the Conference were concerned with extending the concepts of Prolog to include other paradigms such those embedded in "Small Talk" or in "Lisp"

and one delightful experimental system called "Tau" was demonstrated at NTT which allows for the switching of the three paradigms at will by the operator. It was a considerable tour de force by the designer but it wasn't at all clear whether the average user would be able to keep track of the complexities that would be generated in having such a wide range of flexible approaches.

All the basic equipment promised for the first phase has been produced. I will not go into details of such machines here because they have been widely publicised elsewhere. However, in summary, these are; The Personal Sequential Inference Machine operating system, SIMPOS which relies upon a kernel called KLO. A number of demonstrations, using these machines, of a somewhat trivial nature were shown at the Conference. The development of the next generation of operating systems-KL1 and the design of a parallel inference machine are well advanced. A relational database machine based upon a binary relational mechanism has been produced and this also was demonstrated. The application languages that hey are developing are two, there is ESP (a logic programming language with object oriented features), and MANDALA (A knowledge programming language being used to allow for the creation of knowledge representation languages). These represent very powerful flexible tools for the development of expert systems and research into knowledge based systems. However, it was made very clear in discussion that there has been a realisation by the research team that the problems they are faced with are a good deal more substantial than perhaps were perceived at the beginning of the programme. There was a considerable reservation expressed about the way in which applications would be developed and it was widely acknowledged that there is no body of expert knowledge codified waiting for use and a great deal of work will have to be done in order to achieve that.

One notable omission from the initial phase of the 5G programme was work on speech recognition which early press comment had hailed as being one of the major activities of the 5G project. However, it was made clear in the introductory speeches that speech recognition and image processing was considered to be close to commercial exploitation by industry and so the government sponsored programme had decided to leave speech recognition to industry for the moment. The ICOT programme would take up research in the intermediate phase. In visiting various companies, all of whom were eager to demonstrate their speech recognition, those shown were of relatively limited capability. The impression given was that most firms intended to put into the commercial market place these machines of modest ability with a view to opening up a range of applications. Further research would then be done to enhance the capabilities of these machines. However, having said that, there was a very considerable body of work evident on simultaneous translation both between Japanese and European languages and between European languages and Japanese mainly aimed at technical literature or systems manuals. Demonstrations were given which produced very passable translations at the first attempt although inevitably some nonsense statements were made by the machine. A particularly interesting application of simultaneous translation was being undertaken at NTT where they were attempting to translate dramatic

newspaper statements of violent crimes. The subject matter was chosen in order to provide simple language with very clear cut scenarios and strong dramatic and contextual changes so enabling the inference mechanisms and semantic analysis to be undertaken more easily. From the evidence seen these seem to be working remarkably well.

These machines have been developed, the software exists, the first applications are being sought, but what are the immediate future plans? The programme is said to be on course and so the general direction is already given. However, the impression was formed that the detail of the programme was not being expressed as clearly as for the first phase of the programme but nevertheless there are some very impressive proposals being put forward for this next phase. These will be summarised under separate headings:

INFERENCE SUB-SYSTEMS

The parallel inference machine architecture has been designed and a hundred processor prototype for hardware simulation has already been built. The processing element and the multiple element module of the real system has also been built. A software simulation for the 1000 element processor has also been undertaken and some work has been done on interfacing it with the knowledge based machine. However, a major problem area that has still to be investigated is to what degree parallelism is required in many of the problems to be faced by the artifical intelligence community. Many papers refer to calculations of the degree of parallelism required for specific problems but such results as were shown seem to indicate that parallelism of a much more modest level may well be adequate - parallelism of about up to 16 parallel channels. This is clearly a major research area for the future.

KNOWLEDGE BASED SUB-SYSTEM

The target here is to have 100 to 1000 giga-byte capacity with a few second retrieval. They are looking at a number of knowledge based machine architectures including distributed knowledged based control mechanisms and large scale knowledge based architectures, but there are many problem areas to be studied. It was the opinion of a group of European experts who met together under the auspices of the EEC Commission to consider the outcome of the Conference that the knowledge based mechanism being attempted at the moment, while very flexible, may in fact incur very substantial overheads in carrying out practical problems and future generations of knowledge based machines may see higher levels of relational mechanisms implemented in hardware.

BASIC SOFTWARE

The kernel language for the parallel processor of the next generation will be KL1 and so the plan is to build a processor for KL1 together with its support environment and to plan a further generation of kernel language based upon the experience gained with KL1 and KL0. This would be called KL2. On the problem solving and inference mechanism front they intend to work on system methodology to achieve a problem parallelism of 100. This refers to the problems indicated above. They see that an increased emphasis would be placed on co-operative problem solving via multiple expert systems, a technique very similar to the blackboard type of approach to large scale information sifting and analysis problems. They have also expressed an interest in moving to high level artificial intelligence but from the comments made it would appear not to have any special insight into how this should be done, yet exists. However, there is a clear wish to exploit the concepts of knowledge based management and they have a programme to develop knowledge based management software which will have knowledge representation languages for specific domains and knowledge acquisition tools. Time and time again the vast amount of work still to be undertaken knowledge acquisition and codification in order to implement problems on these machines was emphasised.

INTELLIGENT INTERFACE SOFTWARE

There was a feeling that there is still a lot of work to be done on the structure of language and they are planning to develop or continue development of a semantic dictionary and semantic analysis systems, sentence analysis and composition systems, and produce pilot models of speech, graphics and image processing interactive system. Some evidence was put forward that they also intend to tackle the intelligent programming software problem. Although with the the advent of the new software engineering programme, it is not clear how much overlap there would be. But under intelligent programming software they intend to continue with the specification, description and verification system, software knowledge management systems, programme transformation, proof mechanisms, a pilot model for software design, production and maintenance systems. This last item sounded to be very similar to the SPMMS type of project already under way under the Esprit Programme. Attempts will be made to build demonstration systems to show off the the power of these new techniques.

These are a very impressive list of goals and of achievments. They are not however, unique. There was a general impression that the hardware currently available is not dissimilar to that which is coming onto the market at the moment from both European and US sources. However, there is no doubt that the very wide base of technology and trained staff and the ready availability of commercial systems and existing parallel research programmes already under way in industry, represent a very powerful momentum. It is clear that if the technology alone is going to provide the breakthrough in the field of Expert Systems and Artificial Intelligence, then the Japanese will achieve that breakthough. However, they were very eager to ask for collaboration with other national programmes and to suggest that the real problem facing us in the future is the application of these techniques to real industrial problems. In the open discussion at the end of the Conference there was some scepticism voiced by representatives who felt that there was perhaps not a need for this full range of technology for the present state of understanding of Artifical Intelligence and Expert Systems problems. However, one or two speakers - in particular Ed Fiegenbaum - demonstrated that thinking was underway in the US into very deep knowledge based systems which would require processing powers some two orders of magnitude greater than anything being contemplated either in Japan or in the rest of the world at the present time. It is clear that there are problems already formulated in front of us which will require this technology and this represents, possibly, a way in which countries which do not yet have this level of technology and feel that they can not necessarily repeat the research, can in fact become very active partners in this programme. That is by becoming involved in the codification of the expertise which will then be implemented using the machines.

A COMPARISION OF THE ESPRIT AND FIFTH GENERATION PROGRAMMES

In conclusion, therefore, what can one say about the comparison between the Esprit and the Japanese programmes? The Esprit programme was stated to be a reaction by the European Community to the Japanese Fifth Generation initiative. However, the Esprit programme is a very much wider ranging programme, is dealing with a very different industrial infrastructure, is covering a much wider geographical disparity both of national interests and national boundaries and, because of the 50% funding concept builti into the programme, cannot expect to achieve the cohesion that the Japanese programme is achieving. I have indicated that it is my understanding that the EEC are keen that the Esprit programme should become more focused but because the focal point is not being 100% funded, it is difficult to see whether enough industrial goodwill can be generated to achieve this same degree of cohesion as in Japan. If a 50% approach had been adopted in Japan I think it would have failed. The Japanese industrial experience seems to indicate that they have relatively little direct government funding for internal R & D and are much more prepared to invest their own money in "The Way Ahead". They are intensely competitive but this very competition leads to a form of cohesion which, linked to their national characteristics, makes them very competitive. One cannot help thinking that their selective approach to

targets is a much more effective way of proceding than the way in which Esprit is attempting to proceed on a very wide front at the present time. They have already completed a VLSI technology programme, they have the Fifth Generation Programme which corresponds roughly to the Esprit AIP activity, they are planning a software technology programme which sounds remarkably similar to the Esprit Software Technology Programme and of course the Sixth Generation project, announced as being planned to cover research into the structure and function of the brain, if it gets off the ground, will represent something much more advanced than is in the present Esprit Programme but overlaps somewhat with some of the other activities of research in the EEC, but in a more focused manner. Much has been said about the way in which the Japanese are stealing the ideas developed within Europe and the United States but many of these ideas are generated in an atmosphere of academic freedom and the real problem is that the Japanese are not stealing the ideas but exploiting them very much more quickly than is European Industry.

However, on the positive side, there is no doubt that the Esprit project has brought the European I.T. community closer together and industrial links are being formed, as are those between universities and industry, which will strengthen the community in future years. It would be interesting to consider whether an ICOT-like team, or linked teams, would add greater momentum to these areas of the Esprit programme which have not received adequate industrial support. It will be interesting to see what the outcome of the 1985 call for papers will be.

.....

D.G. Morgan
3.5.85

THE ESPRIT SOFTWARE TECHNOLOGY PROGRAMME 1986

THE JAPANESE 5th GENERATION CONFERENCE 1984.

CONTRASTING APPROACHES TO RESEARCH INTO INFORMATION TECHNOLOGY.

D.G. MORGAN

PLESSEY ELECTRONICS SYSTEMS RESEARCH Ltd.

ROKE MANOR.

ESPRIT SOFTWARE TECHNOLOGY PROGRAMME 1985.

- RESPONSE TO 1ST CALL FOR PAPERS DISAPPOINTING IN QUALITY AND QUANTITY

- GREATER THAN 50% OF PROPOSALS REJECTED

- ROLE OF 'A' TYPE PROJECTS.

PLESSEY——— ROKE MANOR

WHY THE REJECTION RATE?

- DID THE ASSESSMENT PANEL APPLY TOO ACADEMIC A STANDARD?
- BADLY WRITTEN PROPOSALS
- PROGRAMME TOO DIFFUSE
- SMALLER FIRMS DISCOURAGED.
- INDUSTRY NOT PREPARED TO INVEST?

PLESSEY —— POKE MANOR

. WHEN PROGRAMME REVIEWED BY TECHNICAL PANEL IT WAS FELT TO BE :-

- TOO VAGUE, TOO SUBDIVIDED APPEARED TOO COMPLEX.
- IGNORING REAL PROBLEMS WHICH WERE SEEN AS :
  - INSUFFICIENT USE OF EXISTING METHODS
  - LACK OF AWARENESS BY MIDDLE MANAGEMENT OF BENEFITS OF S/W TECHNOLOGY
- WEAK ON MEASUREMENT OF BENEFITS.

PLESSEY —— ROKE MANOR

WHEN PROGRAMME RECAST EFFORTS WERE MADE TO :-

- PROVIDE A MORE PRECISE STATEMENT OF THE DESIRED PROJECTS
- INCORPORATE PROJECTS STARTED
- SHIFT BALANCE OF PROJECTS TOWARDS ADOPTION OF S/W ENGINEERING BY INDUSTRY
- OLD MATRIX STRUCTURE DISCARDED THREE AREAS RETAINED
    1. THEORIES, METHODS AND TOOLS
    2. MANAGEMENT AND INDUSTRIAL ASPECTS
    3. COMMON ENVIRONMENT
    4. DEMONSTRATOR PROJECTS.

PLESSEY——— POKE MANOR

SHAPE OF '85 PROGRAMME.

- BASIC TOOLS AND ENVIRONMENTS WERE THOUGHT TO BE WELL-COVERED
- NEW PROPOSALS WERE REQUIRED IN:
  - INTEGRATION OF HARDWARE AND SOFTWARE DESIGNS.
  - ALTERNATIVE METHODS OF SOFTWARE DEVELOPMENT
  - SOFTWARE ENGINEERING FOR SMALL-SCALE, CRITICAL SOFTWARE
  - METRICS
  - MAN/MACHINE INTERFACES.
  - AS ABOVE BUT FOR A WIDER RANGE OF TARGET APPLICATIONS.

COMMISSION REACTION TO JAPANESE 5th GENERATION CONFERENCE.

- REINFORCED CONCERN THAT A.I.P AND S/W TECHNOLOGY LACKED CLEAR CUT TARGETS.
- EFFORTS MADE TO FOCUS PROGRAMME BY LARGER INDUSTRIAL GROUPINGS
- ASKED FOR QUANTITATIVE GOALS (JAPANESE HAVE ANNOUNCED PLANS FOR S/W ENGINEERING PROGRAMME TO CHANGE DEGREE OF AUTOMATION FROM 8% TO 80%)
- INDUSTRY RELUCTANT TO IMPOSE GOALS RETROSPECTIVELY.
- WHAT WILL BE OUTCOME?

PLESSEY —— POKE MANOR

JAPANESE 5* GENERATION CONFERENCE
NOVEMBER 1984.

- CONFERENCE 100% OVERSOLD (USSR APPLIED TOO LATE)
- JAPANESE 5* GENERATION PROJECT HAS FOCUSSED WORLD'S ATTENTION ON INFORMATION TECHNOLOGY.
- HAS BEEN OVERSOLD BY WORLDS PRESS (JAPANESE VIEW)
- OFFICIAL SPOKESMEN EAGER TO DENY ATTACK ON ARTIFICIAL INTELLIGENCE PROBLEM.
- A.I. WILL REQUIRE WORLD WIDE COLLABORATION.
- 5* G. PROJECT ABOUT NEXT GENERATION OF COMPUTERS 'ONLY.'

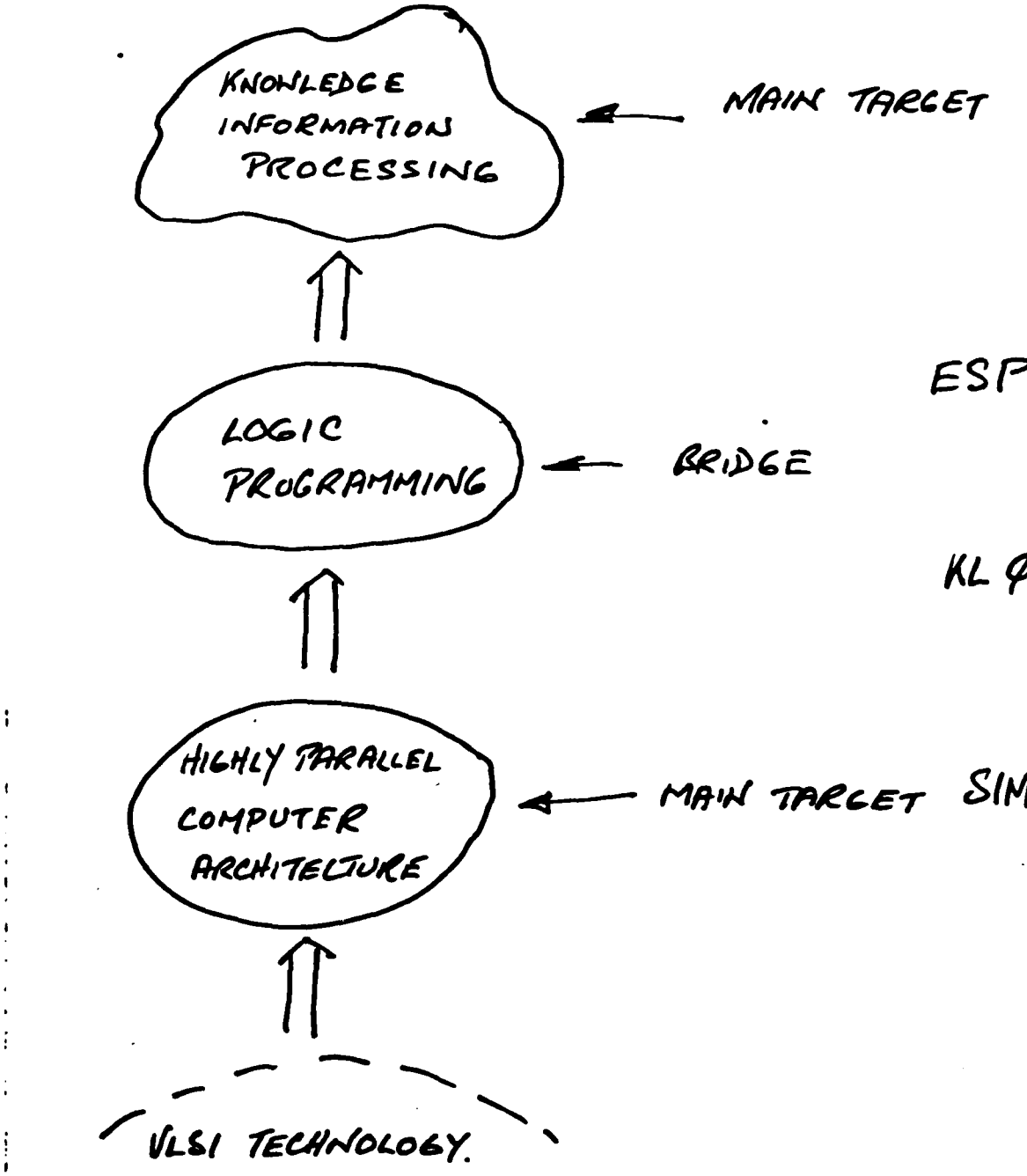
PLESSEY——FUKE MANOR

WHAT HAS 5th G PROGRAMME ACHIEVED TO DATE?

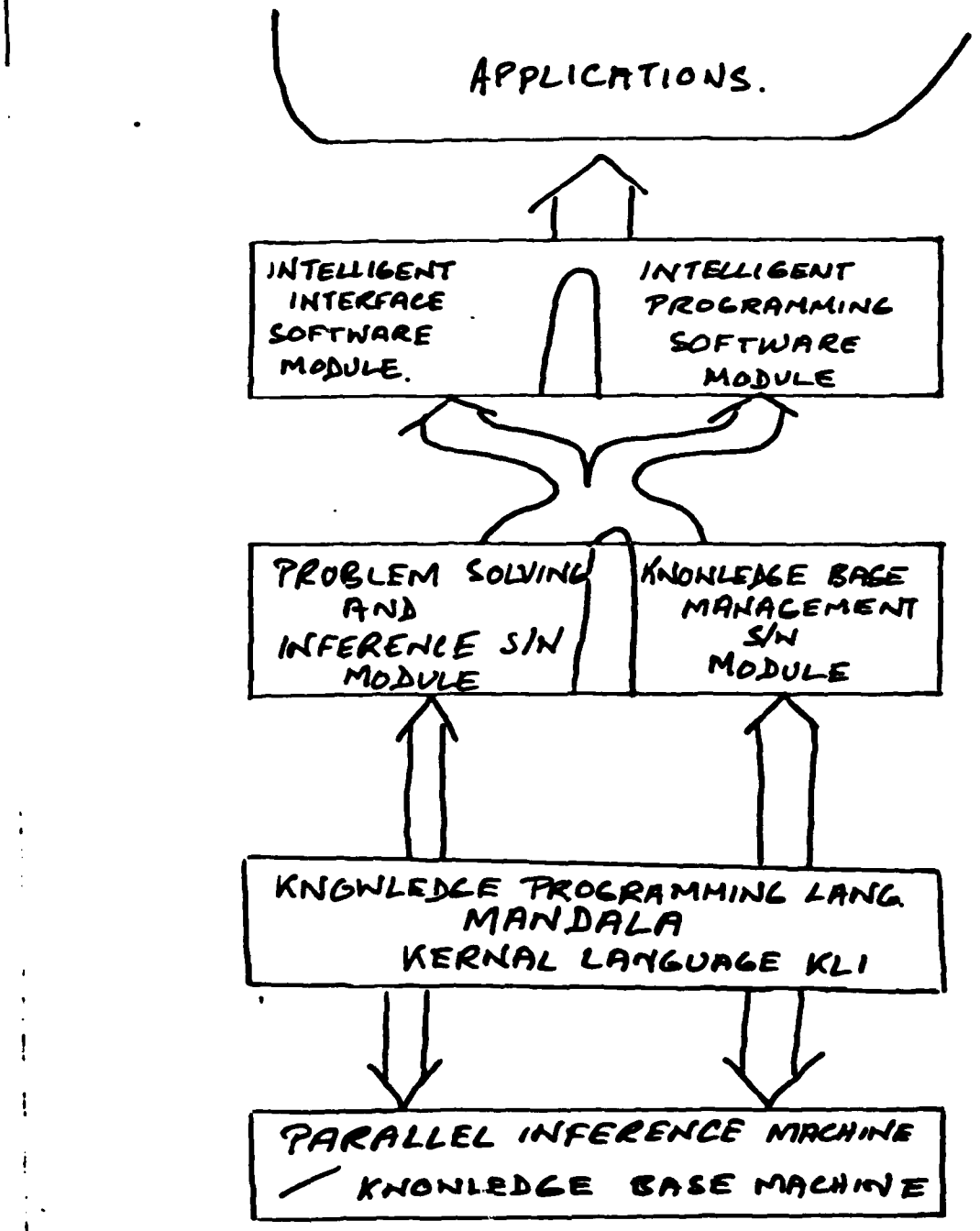
- ALL TARGETS FOR 1ST PHASE MET.
- PRODUCTS ARE TO COMMERCIAL STANDARDS AND HAVE INDEPENDENT COMMERCIAL POTENTIAL
- HAVE A NEW GENERATION OF ABLE SCIENTISTS AND ENGINEERS VERY WELL READ IN THE LITERATURE.
- ICOT CENTRAL ORGANISATION WELL LIKED BY RESEARCHERS.
- ALL CONTRIBUTORS TO ICOT HAVE OWN IN-HOUSE PROGRAMMES AIMED AT IMPROVING UPON ICOT PRODUCTS.
- NTT PROGRAMME INTO SAME AREA PROBABLY BIGGER.

PLESSEY ——— POKE MANOR

FIFTH GENERATION PROJECT.



PLESSEY —— ROKE MANOR



BASIC SOFTWARE SYSTEM.

PLESSEY——ROKE MANOR

SPECIFIC ACHIEVEMENTS

SEQUENTIAL INFERENCE MACHINE
BUILT SIM-P OR PSI

SOFTWARE FOR MACHINE BUILT.
KL ϕ LOGIC PROGRAM BASED.
SIMPOS
ESP ≡ EXTENDED SELF CONTAINED PROLOG

KNOWLEDGE PROGRAMMING LANGUAGE
MANDALA
IN EARLY PROTOTYPE

PARALLEL PROCESSING KERNAL LANGUAGE
KL 1
BEING DESIGNED.

RELATIONAL DATABASE MACHINE
DELTA
BUILT

SPEECH PROCESSING LEFT TO INDUSTRY.

PLESSEY——ROKE MANOR

NAY AHEAD.

- LIFE GETTING MORE DIFFICULT?

INFERENCE SUB-SYSTEMS

1000 ELEMENT PARALLEL PROCESSOR SIMULATED — HOW MUCH NEEDED?

KNOWLEDGE BASED SUB-SYSTEM

AIM IS 100 – 1000 GIGA BYTES WITH FEW SECONDS RETRIEVAL.
— PRESENT STRUCTURE TOO LIMITING?

BASIC SOFTWARE.

SYSTEM METHODOLOGY TO ACHIEVE PARALLELISM.

LACK OF CODIFIED KNOWLEDGE.

INTELLIGENT INTERFACE SOFTWARE.

MAJOR ACTIVITY ON STRUCTURE OF LANGUAGE.

SPECIFICATION, DESCRIPTION ? V & V.

OVERALL TECHNOLOGY NOT VASTLY AHEAD OF US. BUT MOVING FASTER?

PLESSEY —— ROKE MANOR

COMPARISON OF ESPRIT AND 5th GENERATION PROGRAMMES

- ESPRIT MUCH WIDER SCOPE AT PRESENT
- ESPRIT NOT TRUE REACTION TO JAPANESE PROGRAMME.
- FORM OF PROGRAMMES REFLECT NATIONAL CHARACTERISTICS
- JAPANESE "TOP DOWN" ESPRIT "BOTTOM UP"?
- ESPRIT HAS HELPED TO BRING COMMUNITY CLOSER ALTHOUGH MOST OF MAJOR GROUPINGS HAD SOME EARLIER CONTACT.
- FOCUSSED APPROACH APPEARS ATTRACTIVE BUT a) WHO WILL PRODUCE APPLICATIONS?
  b) WHO EXPLOITS WINS.
- WOULD ICOT STYLE WORK IN EUROPE?

FLESSEY —— FUKE MANOR

An Object-Oriented Database System: Object-Base

Herbert Weber

University of Dortmund

Silke Seehusen

University of Bremen

Abstract

An object-oriented database system is a database system in which the concept of abstract data types is used strictly for describing and using the database. In consequence the data are described in conjunction with the appropriate operations, the only operations allowed on that data. The object-oriented database system Object-Base, presented here, allows a hierarchy of data types. That kind of database description enables the system to be adjustable to very different application characteristics. Different levels of user interfaces, from the unexperienced casual user to the sophisticated system programmer, are provided.

TABLE OF CONTENTS

1. Introduction

There are many database management systems (DBMS) available now, and many of them are really good ones. Each of these DBMS is supporting one of the different standard data models (hierarchical, network, relational). Many additional tools, e.g. data dictionary, are provided for managing the databases.

Most DBMS are designed for an administrative or commercial environment. For that application area they are universal. The same DBMS may be used for a university's library and for an inventory of an automobile producer. But the universality has the disadvantage that the DBMS cannot be taylored to each application environment. Such adjustifiability is important with respect to e. g. minimize the effort for application programming on the one hand and with respect to tuning the whole system on the other.

We want to cope with these problems when designing an object-oriented DBMS. Object-orientation has become an important concept in software development techniques.

The concept can be considered only in coherence with data abstraction (/Liskov 75/) and modularization. Thus it enhances comprehensibility, correctness and maintainability of system design and implementation. It has influenced newer programming languages (e. g. ADA; Modula-2, see /Wirth 82/) and operating systems. We want to make use of this concept in database systems as well.

2. Why Yet Another Database System?

Some deficiencies of contemporary data base systems and the following new requirements motivate the newly developed database system.

For in-situ database system support on workstation computers

Many existing commercial applications and many new applications of data base systems seem to need in-situ data base services provided at workstation computers. These computers provide limited memory and processing capacity that may not suffice to support generalized data base management systems of the kind described above.

For application adjustable data base management

More importantly, however, the existing and the new data base applications on these workstations differ considerabely and are thus inadequately supported by only one standardized or generalized type of database system. They are, however, not that different to justify a completely new database system concept for each new application. It, therefore, seems to be attractive to have a database system that can be easily adjusted to many applications with the adjustment much less expensive than a complete redevelopment.
The new type of database system that is going to be described in here is meant to be that kind of an adjustable database system serving for a variety of applications on multifunctional workstation computers.

For different data model support

In-situ database management facilities on workstation computers are different from the more generalized database management faci-

lities in a number of other ways. First of all, they serve as a tool in a narrow application environment with rather specific requirements for data representation and manipulation. In those environments data representation in a form most suitable to the specific application is not only a wish but a very strong demand in order to fulfill the in-situ service requirements. This additional requirement does not only ask for the support of different data representations e. g. of different data models for different application environments. Very frequently the support of different data models (i. e. different views of data) in the same application environment is a must. As a consequence the database system for workstation computers should be adjustable to support different data models in different installations of the system but also in one installation simultaneously.

For semantic data model support

Since the data models supported by the database system are supposed to meet the requirements of particular applications they are usually expected to be semantically richer than, say, the relational model of data. For the support of specific applications semantic data models that enable the representation of static (i. e. declarative) facts and of dynamic (i. e. behavioral) facts are needed to be supported by the system. The system described in here is built to support in that respect.

For Complex Object Representation and Manipulation

Since the system supports a powerful abstraction mechanism arbitrary complex objects for specific applications may be defined and represented in the data base. In addition, the management of the workspace in main memory for complex objects may be simplified with taylored query facilites that do not fetch entire complex object from secondary memory but allow a selective retrieval of components that are needed at the time. This is not to claim that all complex object representation and manipulation

problems for all types of applications are automatically solved with the system. But the claim is made that Object-Base provides all the facilities to adjust the system to different kinds of complex object problems in a flexible manner.

For Long Transaction Management

Some more uncommon database applications require the execution of transactions of very long duration (maybe hours or days). The conventional transaction management schema is not adequate to handle very long transactions. Object-Base treats every transaction as a nested transaction if the data base objects it accesses are composed in a hierarchic fashion. Therefore, the complex object management along with the nested transaction management schema are the prerequisites for the management of long transactions. In addition, transactions only serve as a unit of consistency. Recovery from conflicts and failures in the execution of concurrent transactions are handled with a different schema that does not "undo" partially completed transactions but computes "compensational transactions" on data base objects affected by not fully completed transactions.

For Truely Casual User Support

In addition a new class of users comes into existence with the advent of workstation computers in new application areas. They are primarily interested in the use of prefabricated application programs for a number of reasons:

- They do not have the knowledge and skills to develop their own application programs, or
- they cannot find qualified personel for the development of application programs, or
- they cannot afford to employ the rather expensive program development personel, or
- they cannot spend the time and effort to get themselves edu-

cated in the program development profession, and finally
- they are not interested to get themselves involved in the program development but think of it as a bought in service.

The users of the new in-situ data base systems are also to an ever increasing extent of that nature. They want to run their application programs but cannot involve themselves for the many reasons mentioned above into the development of their own application programs. It was assumed for a while that modern high level - nonprocedural - query languages like SQL would allow so-called casual users to use a data base system. The group of users, however, as it was characterized above, cannot or will not be able to handle a high level programming language that hosts a particular query language and/or the query language itself to write its application programs.
Instead a program generation facility seems to support these users in a much better way than general query facilities. The program generation shall be accomplished through the synthesis of application programs from prefabricated primitive building blocks in much the same way as program generators built programs from prefabricated macros. The group of users mentioned above is meant to be supported by the Object-Base and its associated application program generator as described later on in that paper.

For Low System Overhead

Last, but not least, many of the conventional applications of databases are very static in nature. They are primarily used for the same kind of standard applications over and over again. Additions to the established applications are to occure seldomly, existing applications almost never disappear.
The use of a database systems that supports precompilation and interpretation of queries is not justifyable for this kind of application pattern. The existence of both increases the complexity of the database management system considerably, increase the general overhead and leads consequently to a reduced overall

performance. Systems that only provide a pre-compilation facility are expected to suffice in many cases. The Object-Base is aimed at being used for the stable kind of applications mentioned above giving its user a system of lower complexity and lower system overhead. Because of its adjustable nature the system may, however, be enhanced by an interpreter facility later on easily when needed.

For General Purpose Database Management

Although Object-Base has been developed especially to serve as an in-situ data base tool it may of course also be used as a general purpose data base system that serves as a central data repository for a large community of different types of users. It can serve in that respect if a flexible interpretative query facility is not needed and the application environment only requires infrequent changes of application programs, integrity assertions, access constraints etc. or if an interpreter facility is added on later.

3. Object-Base:A Database and Application-Generator System for Multifunctional Workstations

Powerful workstation computers supporting multi-user operating systems like UNIX are increasingly used in many applications areas like office automation, CAD/CAM, software development etc. All these applications need to be supported by adequate data management capabilities. Since the applications are rather different in nature different data base support function are needed for them. The Object-Base is aimed at being an adjustable database system for multifunctional workstations that can be taylored efficiently towards different applications.
Workstation computers are also increasingly used by non-sophisticated programmers but rather by end-users. They are primarily interested in running pre-fabricated standard application programs. Database accesses are made possible for this type of users through the execution of predefined standard queries.

Object-Base provides means to flexibly combine pre-defined simple queries into complex constructed queries thus enabling the generation of application programs from pre-formulated "query macros". This feature of Object-Base is called Application Program Generator.

Like any other relational database system Object-Base may provide relational standard operations (e.g. SELECT, INSERT) on standard data types (e.g. RELATION). These may be used by the application programs and by sophisticated users.

Each user interface is provided by one or more modules, the object descriptions. An overview of a very simple database is given in figure 1.

Every end-user has his/her own view of the database, his/her external interface s/he is interested in. The external interface provides the end-user with operations s/he may execute with

appropriate parameters.

The external interfaces are thus different to external schemata (/ANSI 75/) of a conventional DBS. An external schema is a view of data. And only standard operations (e. g. SELECT, INSERT) are allowed on the data. An external interface is not meant as an interface for an application program. It is designated to the end-user who rarely calls the operations of his/her external interface within a program.

The description of the data and operations of the database is performed in a modular fashion. The hierarchy of application modules plays the role of a conceptual schema of the database although the operations, normally specified within application programs, are integrated. In figure 1 a simple module hierarchy is depicted consisting of four modules only.

The application modules are based on the so called standard modules provided by the underlying database management system. They include basic operations and data types useful for specifying the application modules.

The different user interfaces are discussed in detail in the following sections.

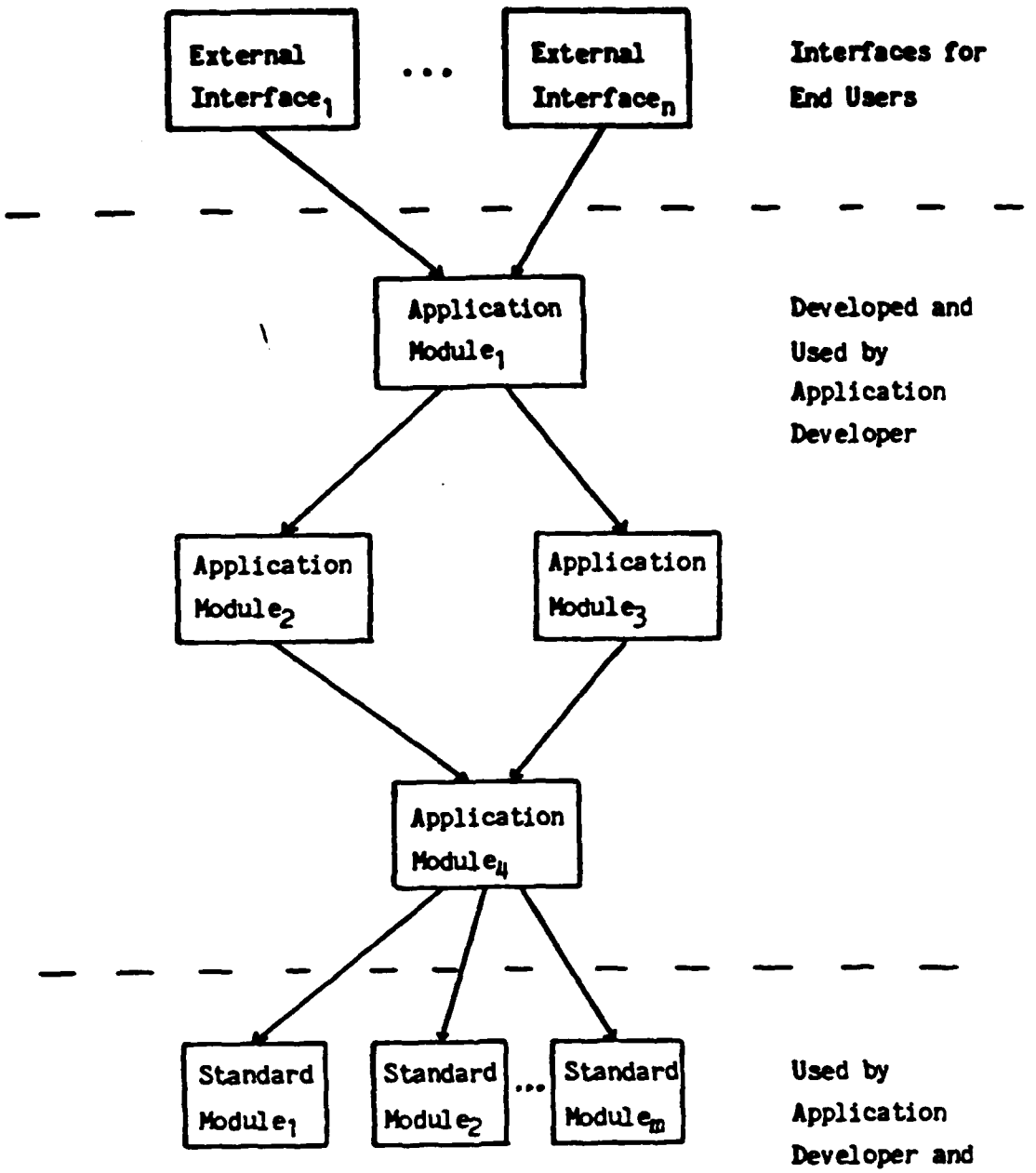


Figure 1: Module Hierarchy of a Simple Data Base

3.1 End-User Interface

Object-Base is meant to support the truely casual user who does not have an interest in programming data base accesses in a higher level query language (e. g. SQL), but whose interest is truely in the execution of existing application programs. He uses the data base system in a "push-button fashion". For that purpose the user will be provided with a structured menu that allows him to select the application program he is interested in. He starts its execution by prompting the identified application program. This can be nicely supported with a mouse and a screen editor if available. For this characteristic the system is also called a "push-button-system".

Each end-user gets his/her own external schema that identifies all his/her private interface entities he/she may want to make use of in his/her access to the data base. The entities identified in the external schema are operation names that identify operations on types of data objects visible for that user. The operation names are associated with placeholders for parameters the user is asked to supply prior to the execution of the operations. After all parameters are supplied properly the user may prompt the execution of an operation.

Each operation name identifies a whole predeveloped application program. An application program may describe a simple query to seek access to one individual data base object or may describe a query that involves access to many different data base objects and computations on those objects. The user, however, never sees the internal structure of the program that is identified by the operation name. He/she also never sees the structure of that part of the data base his/her application program is concerned with.

Example 1:

A forwarding agency for mixed consignment accepts orders from customers who want to send a normally small amount of wares from one city (sender-city) to another city (receiver-city). The agency disposes different wares onto one truck and sends the truck with a disposition list of the different orders on its way.

The user interface of the order acception is a combination of the menue-technique and mask-handling.

The user first chooses the operation wished

Order Acception
Accept Modify Cancel List Info
choose operation:

The user fills in the form with the appropriate Parameters

Order Acception: Accept		
Order No.:		(No. or N for new number)
Customer (Sender):		(Name and Address)
Receiver:		(" " ")
Weight:	... kg	(kg)
Special Issues:		(Text)
fill in forms		

The system provides automatic integrity inforcement. The execution of an application program will only be completed if all integrity contraints that have been declared for data in the database will not be violated during execution of the application program. The system will recognize integrity constraint violations and notify the user of the abortion of the execution of this application program or prompt him/her to either change wrong parameters he/she has supplied to the program or to supply additonal parameters etc. The system, thus, guides the user in the work with the system.
The system also allows only accesses to the data base that are legal for the respective user. The system automatically checks on the access privileges each user has been granted.

3.2 Application Programming

Object-Base supports the construction of the database for a specific application and the associated application programs that embed accesses to the data base. This task can only be accomplished by professionals with a higher degree of knowledge on programming and data base management. They still do not need to be sophisticated programming and data base experts. Object-Base will support the definition of the data base and of application programs to the extent that users of the system in that mode may concentrate on the analysis of the application and neglect details of programming and data base design. This will be made possible through a generator facility that enables the simple construction of data base schemata and application programs. For that purpose the user will be provided with a system supported very high level description and structuring discipline for elementary building blocks of data bases and application programs. For this characteristic the system is also called an "application generator".

In this mode the system provides support for the construction of the data base and of application programs from building blocks through the definition of new building blocks and the re-use of prefabricated building blocks, the standard modules and the already defined application modules. It also supports the formulation of integrity constraints and access constraints in an unambiguous fashion. A syntax-directed editing facility that is not an integral part of the Object-Base may be employed to partially automate the development and validation of the building blocks and their proper interconnection.

The user gets access to an extended conceptual schema that identifies all entities he/she may want to make use of in the synthesis of application programs. The entities identified in the extended conceptual schema are module descriptions (/Weber 83/).

A module description consists of

(I) the description of one type of data object,

(II) the description of all operations that may be applied to the kind of data object described in the module (i. e. prefabricated simple queries).

Example 2:

The orders of the forwarding agency introduced in example 1 are specified. This is a simple version of the module "orders" described in pseudo-code. The module "orders" uses the module "order" managing one order.

```
module orders
module interface
purpose: managing orders
permanent data: ORDERS
operations:  accept (in  sender_name, from_city, receiver_name,
                         to_city: NAME, weight: KG, charge: DM,
                     out Nof: NAT, charge: DM, message:MESSAGE),
             cancel (in Nof: NAT,
                     out message: MESSAGE)
module body
permanent data ORDERS = set of ORDER
operation  accept  (in  sender_name, from_city, receiver_name,
                        to_city: NAME, weight: KG) charge: DM
                   (out Nof: NAT, message: MESSAGE)
    call order_numbers.new_number (out Nof);
    call  order.create  (in   Nof, sender_name, from_city,
                              receiver_name, to_city, weight,
                              charge,
                         out  message)
end accept

operation cancel (in  Nof:NAT, out message: MESSAGE) is
    call order.delete (in Nof, out message)
  end cancel
end module orders
```

Application programs will be synthesized from pre-fabricated simple queries on data objects that have been described in the respective module description.
All the modules the simple queries are taken from to formulate an application program are said to be the constituent modules of that application program.

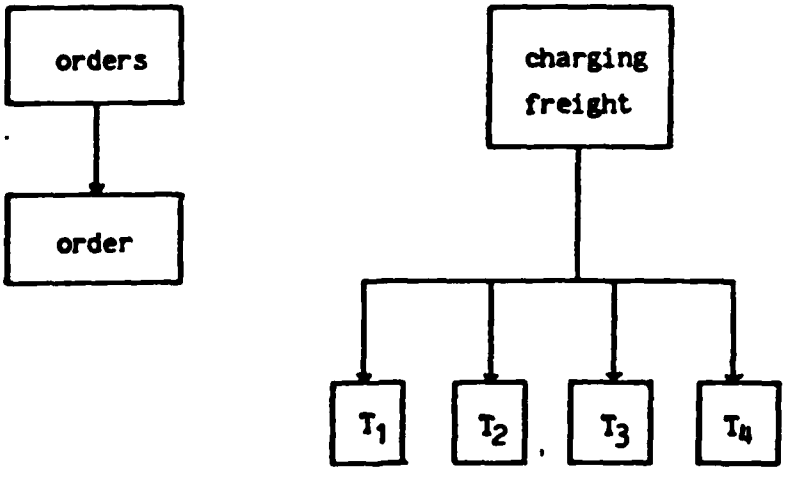
A newly formulated application program will be made a constituent part of a new (i. e. higher level) module with the composition of the data objects of all constituent modules as its new data

object. This features enables the hierarchic composition of application programs from pre-fabricated building blocks. We call the newly formulated module that hosts an application program an application program module (in short: application modles see figure 1). After its formulation it becomes a pre-fabricated module for later use in other even higher level application program constructions.

Example 3:

Assume the forwarding agency of the proceeding examples is already managing a database for orders and one for freight charging including the different tables for freight rates. The end-user uses these two databases seperately. When accepting an order s/he first calls the freight charging program to charge the freight and then fills in the amount in the mask of the order acception.

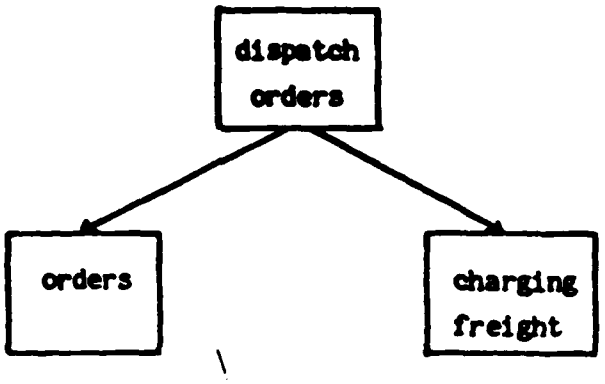
There are the modules:



tables for freight rates

If now the freight has to be charged automatically when inserting an order the interface of the modules "orders" and "charging freight" are used as part of the abstract level language provided

by the already existing module hierarchy:



The operation of dispatching an order would look like:

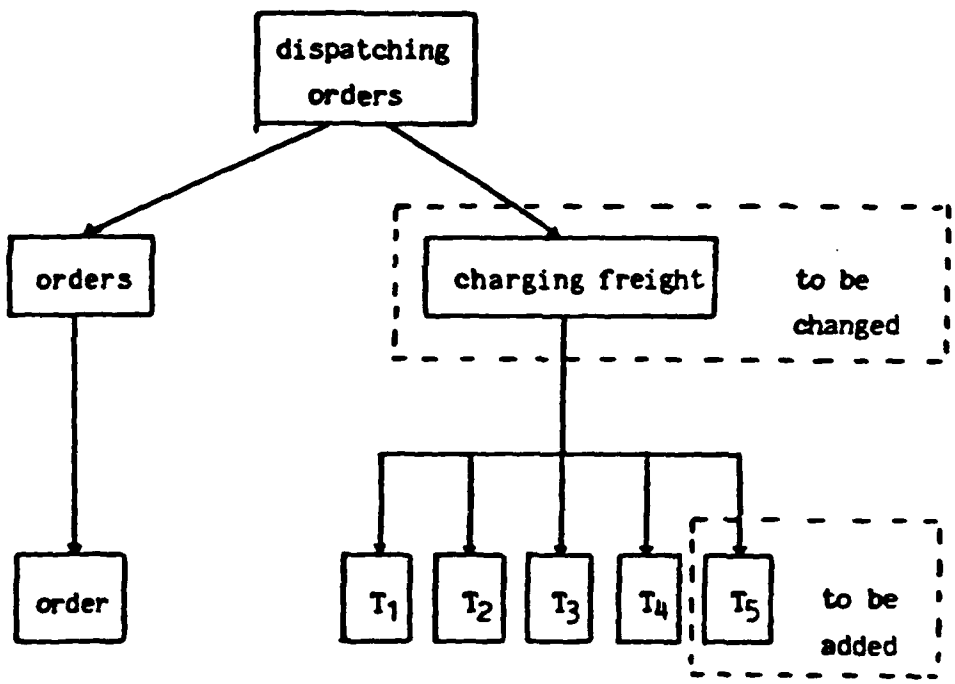
```
operation  accept (in:  sender_name, from_city, receiver_name,
                        to_city: NAME, weight: DM
                   out: No#: NAT, charge: DM, message: MESSAGE)
                   is
   call charging_freight.charge_freight
            (in  from_city, to_city, weight,
             out charge, message)
   if message = ok then
      call    orders.accept    (in sender_name, from_city,
                                    receiver_name, to_city,
                                    weight, charge,
                                out No#, message)
   fi
end accept
```

Since a new service is provided the end-user interface may be enriched now.

Changes in pre-fabricated application programs require the re-compilation of the constituent modules of the application program module that are effected by the changes only. Modules that remain untouched by those changes do not need to be recompiled.

Example 4:

If no new service has to be created but e. g. the freight charging itself has to be changed because of a new table of freight rates, say T_5, has to be added. Then a new module, T_5, encapsulating the new table has to be inserted in the module hierarchy and the module "charging freight" has to be changed accordingly. Because the interface of "charging freight" remains unchanged no other module has to be changed. An end-user would not notice this kind of maintainance.



Application programs may also be constructed through different additional combinations of operations defined in pre-fabricated modules. To enable application program synthesis through additonal combinations of operations all modules that participate in the new combination are needed to be re-compiled. All unchanged modules do not need to be re-compiled.

New application programs may also be built through the addition of a module to the data base and through the replacement of a module already existing in the database system. New modules can be added by only compiling the new module and linking it appropriately to the set of existing modules. The replacement of modules also requires the compilation of the new module and the proper decoupling of the ancient module as well as the proper linking of the new module.

3.3 Basic Interface: Relational Query Interface

In addition to the two interfaces mentioned above a third one is provided that offers a truely relational query language for the retrieval and change of data. The interface is somewhat subordinate to the other interfaces in the sense that it is meant to provide access to the basic data base objects. All user-oriented types of data visible in the external schemas or in the conceptual schema are ultimately represented by unnormalized relations. Anomalies in change operations will be avoided by tayloring operations to each individual relation rather than by decomposition programs by using appropriate primitives of the relational query language. Integrity constraints and access constraints, however, may only be defined for individual base relations. Object-Base used in this mode inforces only these kinds of constraints and not interrelational constraints. It therefore provides only in part what is now frequently termed "integrity subsystem".

Using this interface Object-Base acts like a conventional relational data base system. The data base appears as a collection of base relations and querys may be formulated in a relational algebra type query language. The use of the system in this mode is reserved for sophisticated data base and programming professionals.

The system allows the definition of integrity constraints and access constraints for the data base and of application into Normal-Form Relations and the application of universal query operations. The user of the relational query interface will, however, not be aware of the existence of taylored operations. S/he uses the relational query language in its conventional way. The Object-Base is capable of selecting the appropriately taylored operation on the basis of scope information supplied with the relational query.

The third user-interface is not necessarily restricted to offer relational query processing. Other data models may be supported instead or simultaneously with the relational data model. This is made possible by the earlier described module replacement capability of the system and by the encapsulation capability of each type of data base object mentioned before.

4. General Architecture of Object-Base

Object-Base is a multi-user system that allows a number of users to concurrently use the system in different usage modes (see chapter 3). The system is entirely strucured of separately compilable modules. The modules exhibit a data encapsulation property that enables a rather flexible reconfiguration of the system through additions, removals and replacements of the modules in the system structure.

An overview of the Object-Base is depicted in figure 3. The main components are described in detail later on in this chapter.

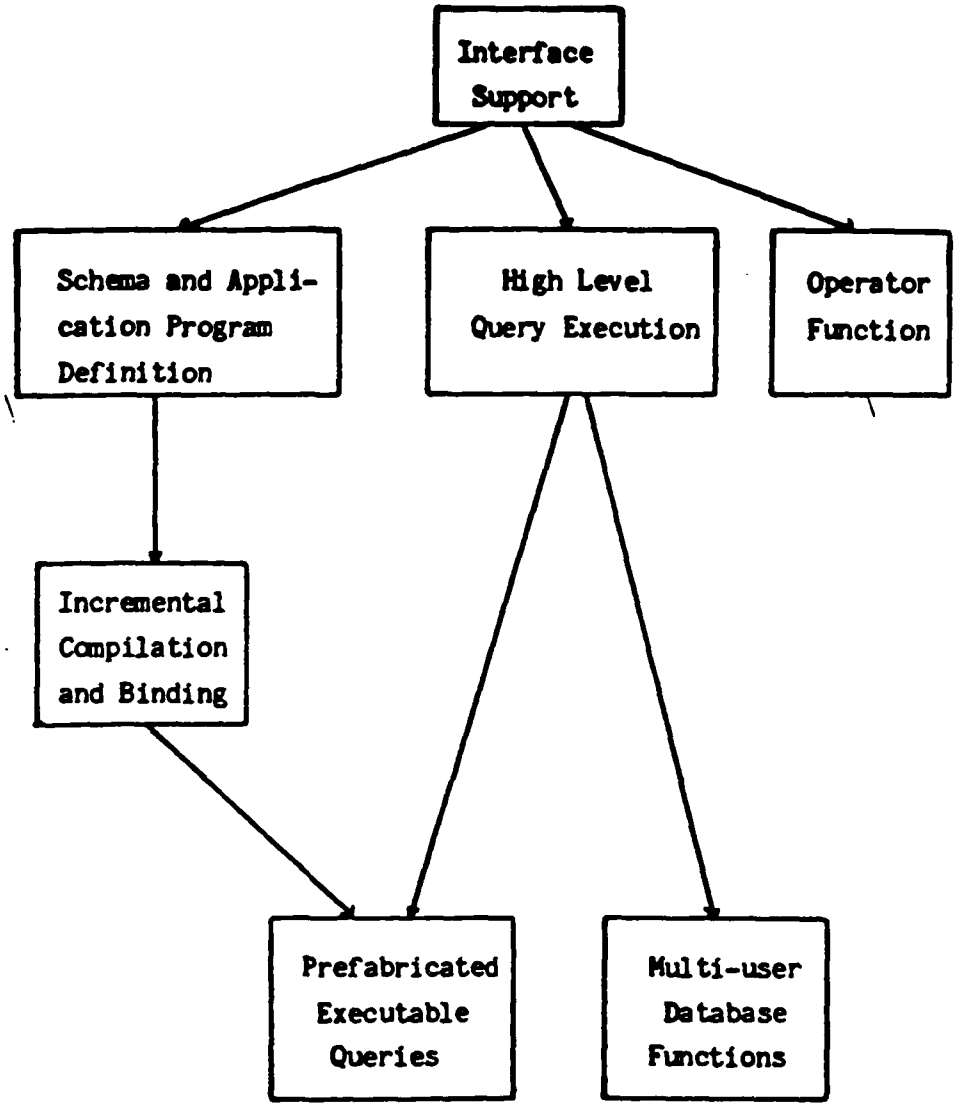


Figure 3: Gross Object-Base System Architecture

The only access to Object-Base is via the Interface Support. It provides the different user interfaces described above. Additionally but nevertheless neccessary is the interface for the operator. The Interface Support checks the identity of a user and gives her/him access to only those operations s/he is allowed to.

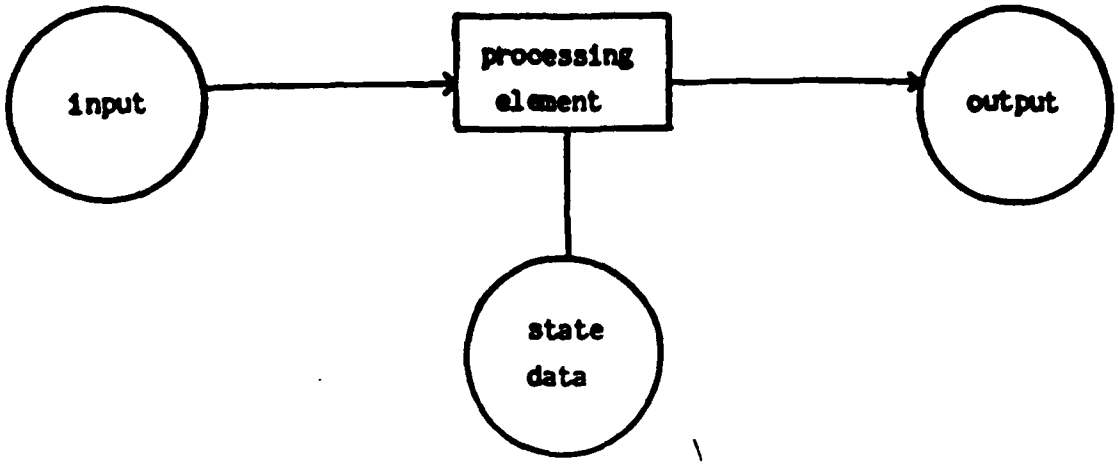
The Operator Functions include the operations for starting, stopping and initiating maintainance operations, e. g. recovery operations. All these operations are neccessary for running the database system.

The Schema and Application Program Definition supports the tasks of database administration and application programming. Facilities for defining, adding, deleting and changing modules are provided. New and changed modules are compiled and connected to the other modules by the Incremental Compilation and Binding. The executable code of the modules is managed by the component called Prefabricated Executable Queries encompassing all programs of executable queries of the database.

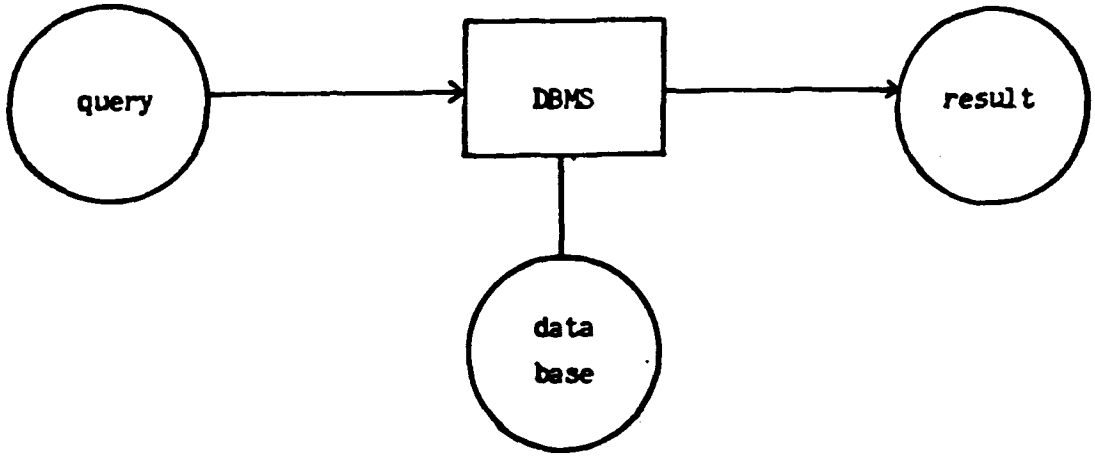
When a user initiates via the Interface Support the execution of a query this query is executed by the High Level Query Execution. Only this component has access to the database via the Multi-User Database Functions. The database functions encompass all functions accessing the database, which are neccessary in a conventional database system as well.

As outlined before the facilities provided by the system differ from conventional system concepts. The differences will be described below.

The general notation used here to describe and depict the system functions is as follows: A system function is defined as a processing element with its associated input and output and in addition with the state data that are subject to changes during the execution of the processing element. This may be depicted as follows:



We may look at the data base system itself as an example for the application of this notation.



The processing element "DBMS" takes a query as an input and produces a result and may change during its execution the state data called "data base".

This description schema will be used to describe the main functions of Object-Base.

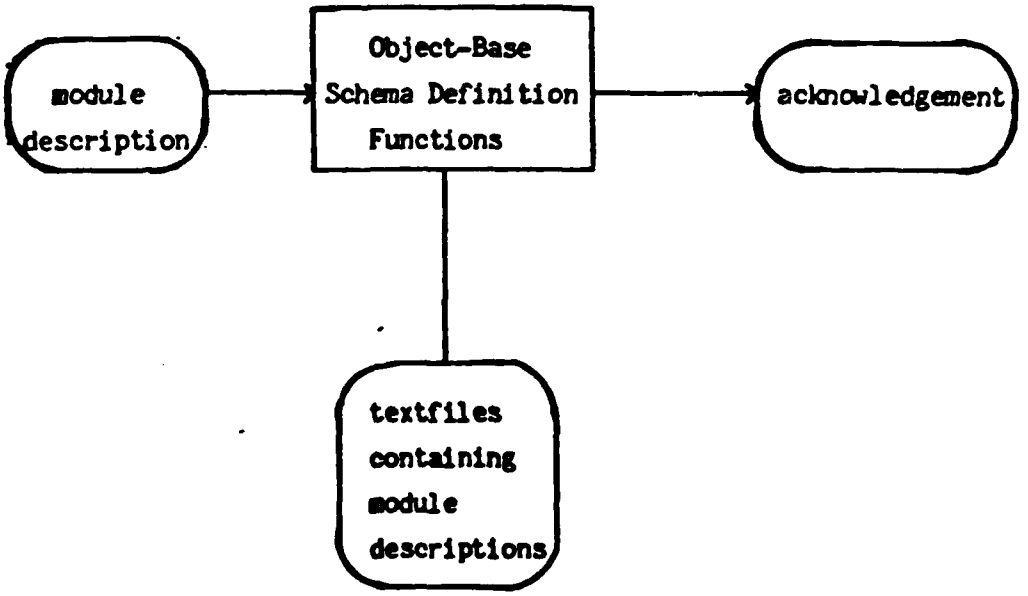
4.1 Schema Definition and Application Program Generation

Schemas serve in Object-Base as in other systems as data descriptions. In addition to data descriptions schemas also encompass

descriptions of all operations applicable to each type of data described in the schema. A schema, thus, contains not only structure descriptions but descriptions of modules each consisting of a data type description and the description of all operations associated with this type of data. This feature supports later on in the formulation of queries and application programs across data units by combining associated operations of different modules into higher level operation and ultimately into application programs.

The data definition language enforces the definition of elementary building blocks of application programs along with the data they are referring to in module descriptions. The composition mechanism that is provided with the data definition language enables the construction of modules in a hierarchic fashion. This allows the definition of arbitrary data compositions in each schema and the construction of application programs out of elementary queries associated with these data within a module.

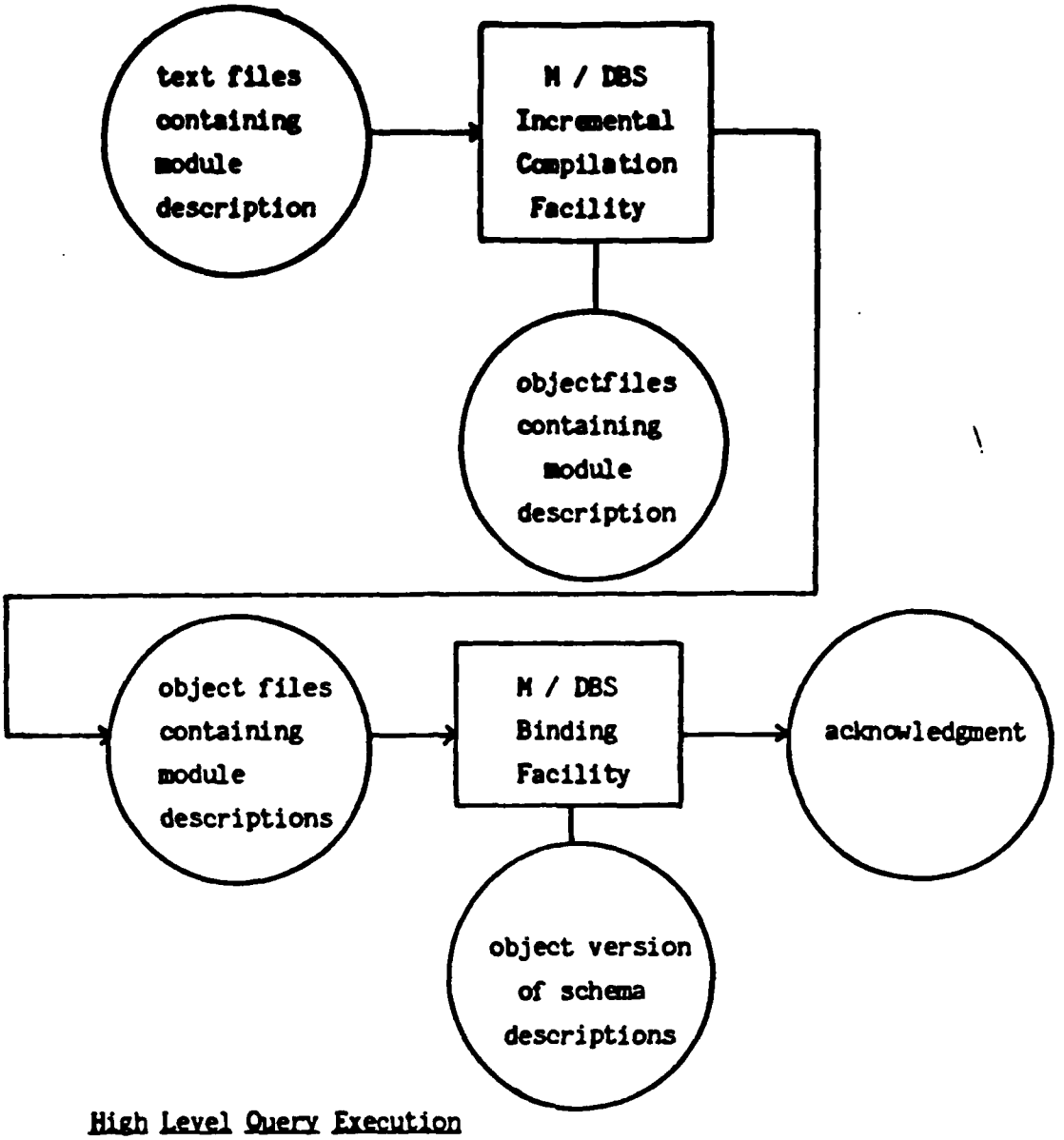
The schema definition and application program generation function may be depicted as follows:



Incremental Compilation

After the definition of a data base schema in terms of modules and after the hierarchical constructions of modules the resulting module hierarchy may be compiled with the compilation function provided in Object-Base. After its compilation the schema consists of executable modules, i. e. the compiled schema contains now object versions of the data descriptions and object versions of the application programs. The execution of the compiled application programs may be initiated through the query processing function provided by the system (see section on query processing function).

The compilation function consists in fact of two subfunctions: one that compiles individual module descriptions and a second one - the binding function - that links a newly compiled module into the already existing module hierarchy at its proper place.

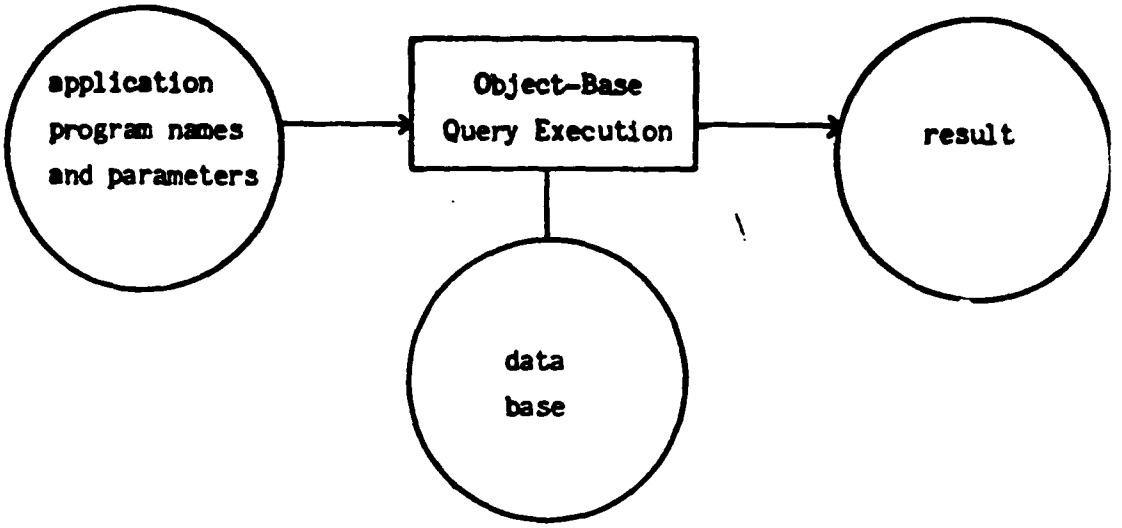


High Level Query Execution

The system provides high level query facilities to support a variety of different users with their own external schema and their own collection of application programs associated with their external schema.

Queries are not formulated by the end-user but rather by a data base expert at schema definition time (see section on schema definition). They may be formulated at a level of abstraction most suitable for a specific application. Each external schema represents, therefore, an application taylored query interface.

The user only initiates the execution of a precompiled query at his/her external schema by supplying the names of the applications programs and its associated parameters.



Multi-User Data-Base Functions

Object-Base enables the concurrent execution of compiled application programs. The consistency of the data base in the concurrent execution of application programs over common data will be guaranteed since each simple query program and composed application program (see section on the Application Programming Interface) is designed to guarantee the consistency of the associated modules' data type.

This in fact results in a new transaction concept and requires synchronization to be orginized in a modular fashion. Provisions for the modular synchronization of concurrent access to the data base are built into Object-Base. This function is a sub-function to the previously defined Higl-level-Query-Function whose services will be used whenever multiple application programs are needed to be executed concurrently.

Since workstation computers are frequently interconnected in

local area networks to enable distributed computing, data management services are needed to be distributed as well. Provisions are made in Object-Base to extend the system into a distributed database system.

5. Status of the System

A first version of Object-Base has been developed at the University of Bremen (/Weber 84/). The development team consisted mainly of students working on the project in partial fulfillment of the curriculum requirements for a Diploma degree.

The project lasted for two years and lead to the specification of the entire system and to the implementation of a rudimentary version of it (running on the VMS operating system on a VAX 11/750). The completion of the system is planned. The purpose of the first version is to act as prototype and so to verify the statements that, hopefully, have been made plausible in this paper.

6. Summary

Starting with the concept of abstract data types a new kind of database system, Object-Base, is developed. The concept turns out to be very fruitful as it leads to many advantageous features of Object-Base.

- The system is adjustable to very different application characteristics. Thus it also supports in-situ database facilities on workstation computers.

- Complex objects are composed hierarchically out of simplex objects. High levels of semantic data models can be provided.

- For truely casual users a push-button interface can be integrated into the application module hierarchy constituting

the conceptual schema with the operations allowed on the data.

- The schema description can be modified easily by addition, deletion and replacement of components (modules).

Because of the object-orientation of Object-Base we think the system to be relevant for the development of future database systems.

References

/ANSI 75/ Interim Report ANSI/X3/SPARC Groups in Data Base Management Systems, FDT-ACM SIGMOD Bulletin H.2.

/Liskov 75/ Liskov, B., Zilles, S. Specification Techniques for Data Abstractions in: SIGPLAN Notices 10,6, June 1975, pp 75 - 83

/Weber 83/ Weber, H. Object-Oriented DDBS-Design in: ICOD 1983, Cambridge

/Weber 84/ Weber, H., Seehusen S. Entwicklung eines modularen Datenbankverwaltungssystems (M/DBVS/), Abschlußbericht, Projektgruppe DBVS, Technical Report 8/84, University of Bremen, Informatik (in German)

/Wirth 82/ Wirth, N. Programming in Modula-2 Springer Verlag, Berlin, 1982

Structuring mechanisms in distributed systems

Radu Popescu-Zeletin
Hahn-Meitner-Institut for Nuclear Research
Glienicker Str. 100
1000 Berlin 39
Germany

Abstract

The paper reviews some of the basic structuring mechanisms in distributed systems from the perspective of an wide-accepted reference model (ISO/OSI). The acceptance of the model and its related standards by international bodies (ISO, IEEE, CCITT, IEC etc.) and the computer manufacturers will have an important role in the development of distributed systems based on available products. The paper outlines the general framework and the requirements of distributed systems and focus on the practical and conceptual problems in using the ISO/OSI in the design of distributed systems.

1. Introduction

Not very long ago, the interface between the user's device and the modem was generally regarded as the line dividing data communication and data processing. The past decade the network has crept through that connector and has infiltrated in terminals, main frames and frontends.

The user needs and a growing market of data communication have precipitated this invasion by calling for increased connectivity, higher reliability, lower costs in networking, support for interconnecting heterogeneous devices and development of systems and applications dealing with communication /1/.

The result of this invasion is a large variety of products, architectures and systems for distributed applications. Recognizing that the user and the manufacturer community have a difficult task to get unscathed through the jungle of concepts and products, different standardization bodies have developed reference models for coherent development and integration of concepts and products in the field of data communication.

The paper outlines the major structuring techniques and analyses the present output with respect to distributed systems requirements.

Chapter 2 gives a short overview on the rationales of the ISO-OSI model and the used tools for hierarchical structuring.

Chapter 3 outlines the main structuring mechanisms and their rationales.

Chapter 4 describes from a personal point of view the state of the art and what is needed and missing with respect to distributed systems.

2. The Reference model for distributed systems

One of the major problem of distributed systems is their inherent complexity. This complexity is motivated by the combination of the

traditional fields in informatics: data processing and data transmission.

The close assembling of the two fields to form distributed systems despite advantages like: increased availability, parallelism, increased reliability and performance, requires the reconsideration of the problems and solutions in both fields. The aim is a distributed system architecture where data-transmission and data processing are melting in one.

The complexity of distributed systems is motivated by:

- If the traditional data processing systems are designed for a certain configuration, the use of DP-systems combined with data networks permits a large variety of configurations to cover a large variety of applications.

- If the traditional tele-communications networks offer highly specialized services (teletex, videotex, terminal access etc.), the distributed processing systems are required to perform a variety of functions modelling a growing field of applications and requirements.

- If the traditional data-processing is based mainly on sequentiality, the natural starting point for distributed systems is parallelism. Parallelism introduces a new way of thinking in problem formulation and solving.

- Expected advantages of distributed systems like: higher availibility, reliability and performance are paid internally by complex algorithms which are characteristic for distributed control of resources /2/.

The well-known technique for solving complex problems is the decomposition of the initial problem in a model of less complex problems also known as the architecture of the system.

At international level different standardization bodies have tried to develop systems architectures with reference character for complex systems. One example is the ANSI/SPARC model for database systems.

For distributed processing systems the international efforts have merged in the well-accepted ISO-Open System Interconnection Reference Model (ISO/OSI).

It is probably interesting to analyse the term: Open System Interconnection Reference Model, because it hides the aim of the model. The scope is to interconnect systems which are open in their architecture to co-exist and co-operate with other similar systems. The key issue is a common architecture which is flexible enough to allow changes without throwing away the whole system.

The method consists in describing a model of a network of Open Systems (necessary for OSI standard designers) as a network of models of Open Systems (necessary for implementing an individual Open System).

The second characteristics is the reference character of the developed architecture, which allows to develop components and products with reference to an accepted reference model.

That means that the reference architecture must provide a precise frame-work in which the internal functionality of Distributed Systems is clearly decomposed in independent components which encapsulate functions without influencing each other.

The precise formulation of the model address the external visibility of each component (layer) and the relationship between the components of the model.

This technique allows products development for the different components in a disciplined way.

The modelling technique of encapsulating related functions in modules and providing autonomous external visibility is closed to the well-known concepts of abstract data types. The resulting architecture is a layered one which allows as well the discrete evolution of hardware and software in the different layers as the introduction of new systems without making previous implementations obsolete.

The reference character of the OSI model has already and will have a dominant role in future distributed systems since it imposes a rigorous discipline by providing a logical frame-work for a complex domain and for the development of products and systems to achieve compatibility.

The adapted layering technique is characterized by two major concepts (fig. 1).

- The layer service: which is the set of capabilities offered at the boundary of a layer to a user in the next higher layer. Note that the service is an abstraction by which the capabilities offered by a layer (in using all lower layers) are specified and that the service definition is independent of any particular implementation.

- The protocol which defines the rules of interactions between the entities which are situated in the same layer but pertain to different systems.
  It is obvious that by the definition of the service offered by a layer and the definition of the service offered by the lower layer the functionality of a layer is completely specified. Note that different protocols may carry out this functionality.

The result of the conceptual analysis of distributed systems functionality is a seven-layer architecture. One of the key issue of the model is the separation between data-transmission and data-processing domains at the boundary of layer 4.

The first four layers provide an uniform communication kernel and deal with functions necessary to provide and support a diversity of topologies, error recognition and recovery mechanisms, efficient transmission and costs optimizations.

A firm end-to-end basis for inter-process communication in which all the above problems of data transmission are hidden for the user is an essential feature for the development of distributed systems.

Note that in this chapter we do not address the different products and standards in the different layers of the ISO-OSI their advantages and their weak points but the rationales for developing a reference model and the model itself.

The upper three layers are probably the most interesting since they address a domain which was influenced mainly by the traditional operating systems and basic constructs of high-level languages.

OSI brings new influences in these functions focussing on communication and dialogue aspects which have been poorly treated in the past.

Fig. 2 focus on an informal comparision among functions offered by operating systems and programming languages on one hand and functions offered by the Session and Presentation layer as described now in the standards /2/.

3. Structuring criteria in distributed systems

Basically there are some general criteria for structuring distributed systems. These criteria are:

- Space Structuring
- Time Structuring
- Data structuring.

The structuring process in the distributed system design is depicted in fig. 3.

3.1 Structuring in space

One of the important structuring criteria of a distributed system is its structuring in space. By structuring in space we do not mean a mapping of the system in a certain topology but rather a logical structuring in autonomous components. It is important to underline the independence of the structure from the topology since this is necessary for the open characteristics of the system.

It is also important to perform first a horizontal structuring in space in components at the same level encapsulating functions from a top-down design approach. The vertical space structuring will then define the necessary functions in the layered architecture. Note that OSI provide till now only vertical structuring and only very few provisions for the horizontal one (MHS,JTM).

It is clear that the actual developed standards in OSI and the related products model co-operation between autonomous systems and provide a framework to bridge mainly distances and heterogenity.
The horizontal space structuring is necessary for a distributed application where components are tighly communicating to provide one single application. In this class of distributed systems are the fault-tolerant systems, resource sharing systems, real-time applications and systems designed for high-reliability and performance. The space structuring must provide horizontally in each layer explicit address-spaces, which are governed by explicit protocols.

3.2 Time structuring

In order to be able to cope with the parallelism in distributed systems an important aspect is the structuring in time of different activities. This allows not only to exploit the distribution by allowing independent, parallel processing but also to define the best software and hardware topology.

Tightly coupled with time structuring are aspects like synchronization of parallel processing, operation, atomicity and data consistency.

Again, the related ISO-OSI standards provide at this time only mechanisms for co-operation of point-to-point autonomous applications and the user of the model has to deal with more sophisticated communication aspects outside the model.

A very important point is the requirement of tools for validation and verification of the correctness of the protocols and possible sequences of events in a complex environment. The availability of tools has to accompany the development of the distributed systems.

3.3 Data structuring

From the experience gained in the last years, one can classify the different structuring approaches of a distributed system in two gross classes:

- Program-oriented
- Data-oriented.

Comparing the two approaches it seems that the data-oriented one has a lot of advantages. By a precise definition of the data structures in each level of abstraction we gain clarity in the distribution.

The OSI-model provides a clear separation of the protocol elements pertaining to a certain level and transparent data from/for the level above. This supports the independence of the layers in the model. The use of abstract data types concept to model the different levels and components in a distributed system is appealing in this context.

Tightly coupled with data structuring principles are error-detection and recovery mechanisms. Walker made already 1977 the observation:

" For a large majority of applications, it is much more cost effective to expect failure to occur and to recover from them than to aim for a totally fault-free system."

The encapsulation of errors in structural components allow the development of error recovery mechanisms and though provides a higher reliability in the systems. The developed products and standards in ISO/OSI follow already these principals.

A good illustration is the design of the transport protocol classes 1 and 3 where the life time of a transport connection may span more than one network connection using recovery mechanisms from network failures.

4. State of the art

The development of standards and the products within the OSI model have been influenced by the immediate market. They mirror a certain class of applications in the field of office automation and telematic services.

The field of applications which is now covered by the standards have the following characteristics:

- The systems involved are autonomous (horizontal space structuring - same functionality in each system).

- The applications in autonomous systems co-operate in point-to-point communication regime.

- The different standards for each level have been developed to bridge distances and heterogeneity.

Fig. 4 depicts the standards for each level

4.1 Open systems

The term open-systems hides the wish that each system will conform to the reference model and will be able to communicate with all other systems. Since nobody can forsee all the future applications to be supported by these systems the wish is quite utopic. It is interesting to observe which are the mechanisms to support this wish.

The most important one is the provision of negotiating at service, and consequently at protocol level the required quality of service. Note that all standards at the different levels provide this capability.

The quality of service negotiation is specific for each level since each level is designed to perform a certain functionality and is represented by entities in a certain address space.

Note also that the negotiation takes place between at least three partners: the two users of a service and the layer below and in that negotiation all three partners may intervene. The result is an agreed communication quality during the data transfer phase. The quality of service is not the only subject for negotiation. Since powerful systems may communicate with weak systems the negotiation includes the set of primitives, subsets and functional units supported by the two open systems in order to establish the common denominator in communication.

That means that a system in order to be an open system has to fulfill at least some minimal requirements and that all systems have to be designed in such a way that they can degrade their functionality if necessary down to a well-defined minimum.

A flexible negotiation and the ability to degrade to a level of functionality imposed by the peer system is a new quality in software and hardware product design. The negotiation rules of upgrading/degrading a system are stated as general conformance rules for each level.

4.2 Protocol Engineering

The above described techniques have been successfully applied in the design of standards and their associated products in the ISO-OSI environment.

A new discipline has evolved which from literature /4,2/ is known as "protocol engineering".

The term new is as always relative but it is important to note that it is the only field in which:

- a general model was developed and finalized before specific standards and related products have been produced

- the development of the model and its related standards and products have been accompanied by the development of formal description technique, certification and validation tools.

The domain of "protocol engineering" is depicted in fig. 5.

The development of standards for the distributed system follow in their implementation some very precise steps, which mirror on one hand the modularisation and hierarchical structuring and on the other hand the involvement of different protocol engineering techniques and tools.

The wide acceptance of the ISO-OSI model and concepts by international organizations like CCITT, IEEE, ECMA, IEC and of the different product suppliers is an important hint for the relevance of the model in the future.

4.4 Some conceptual and practical problems

There are a number of general issues which are not supported or not clear in the ISO-OSI reference model specification and its related standards. These issues appear when the network designer intends to develop implementation specifications which are required to conform with the standards and at the same time to offer practical solutions for the network design /5/.

1. Layer independence

   As already mentioned the adopted layering technique is characterized by two major concepts:

   - the service definition describing the external visibility of a certain layer and

   - the protocol definition describing the internal functioning of the layer.

   Although these architectural concepts aim to layer independence the services defined in the related standards for each layer do not provide completely this goal. This is mainly due to the fact that the services are defined as a three party communication. The three communicating entities are the two users of the service and the layer below as service provider.

The implication of this fact is that the specifications of the services provided by each layer define the capabilities offered by the layer and the behaviour of the two users of the service in the layer above.

A neutral service definition without the involvement of the behaviour of the entities in the layer above is probably the only way to achieve the aimed layer independence.

2. Multicast communication

The ISO-OSI model provides provisions for point-to-point communication only. A large class of applications requires multicast communication regimes (distributed data-bases, mailing and tele-conferencing systems etc.). At the present time the user of the model has to solve the communication aspects for multicast applications outside the model. Multicast communication introduces not only new addressing capabilities but also synchronisation mechanisms to preserve the consistency of data, and the atomicity of operations which are poorly treated or completely missing in the actual proposals. These aspects are essential for a large variety of applications.

Since one of the aims of the model is to relieve the users of the model of communication-oriented aspects, for this class of applications the model failed its aim.

3. Layer entities

Another practical and architectural problem is the fact that in the OSI Reference Model the existence of entities in the next higher layer is assumed at connection set-up time (e.g. Session Entities for the Transport Layer in the establishment phase). This assumption is not true for most implementations, if the implementator cares for implementation efficiency and so does not implement a multitude of dummy processes which have to wait to be activated. That means that before a CONNECT indication may occur the creation of an entity in the Session Layer has to be performed.

It is not clear from the service specification which entity in which layer has to enter the termination phase if the entity cannot be created or when a deadlock situation occurs.

4. Dynamic change of the quality of service

The present documents specify the negotiation of the service quality only at connection set-up time. There are many cases where a dynamic change of the quality of service is required. In the existing documents a quality of service may be changed only by involving the termination of the connection and then by re-establishing a new connection with the new service quality. This schema is too rigid to too costly.

A practical problem in the network design is also the criteria choice on how different quality of services at different layers can be mapped.

5. Quality of service parameters

Some of the quality of service specified in the ISO draft proposals as parameters of the service primitives are difficult to interpret if not impossible to support. For example, the meaning of the negotiation of the connection establishment failure probability at the connection set-up time or of the DISCONNECT failure probability is not clear for the user of the service. Other parameters can be supported only in conjunction with a powerful network management procotol specific to the layer offering the service.

The requirement to guarantee a certain quality of service is mandatory for a large class of applications (e.g. real-time). The present standards and products cannot ensure a quality of service during the life-time of a connection. The layer providing the required quality can only "do its best" without guarantee.

6. Performance

Very often the network designer is faced with the question "Does the implementation of the seven layers affect the network performance and if yes how much?"

The answer of the question is not easy, because the pilot implementations build a network operating system on top of the existing operating systems in each host. The scope of the standards in the OSI-environment is that the computer manufacturer has to integrate these features in their operating systems for communication purposes.

The integration process has already begun and many manufacturers provide already computers and workstations with OS architectures based on the OSI-standards.

Conclusions:

The ISO-OSI model is an efficient tool for the development of the distributed systems even if does not cover all aspects required by distributed systems. It is a good begin since by adopting the same reference terminology and design concepts minimizes the understanding overhead.

On the other hand since the manufacturers have adopted the model for their product developments a large compatible product variety is expected in the near future relieving the user of own solutions. Wide-accepted structuring mechanisms of distributed systems and the necessary tools to specify, validate and verify the design are mandatory in distributed system development.

References:

/1/ Mier, Edwin (1982): High-level protocols and the OSI reference model, Data Communications.

/2/ Zimmermann, H.: On protocol engineering to be published

/3/ Piatkowski (1982): Protocol engineering, Proc ICC Boston.

/4/ Popescu-Zeletin, R. (1983): Some critical considerations on the ISO/OSI RM from a network implementation point of view IEEE Proc. Eights Data Communications Symposium, Cape Cod.

Figure Captions

Fig. 1: Services and protocols in ISO/OSI

Fig. 2: Analogy OS/Prog. languages and ISO/OSI
(from H. Zimmermann "On Procotol Engineering")

Fig. 3: Structuring mechanisms in distributed systems

Fig. 4: Standards supported by different organizations
(from DATACOM 2/84)

Fig. 5: The protocol-engineering domain

Fig. 6: Hierarchical development of a distributed system.

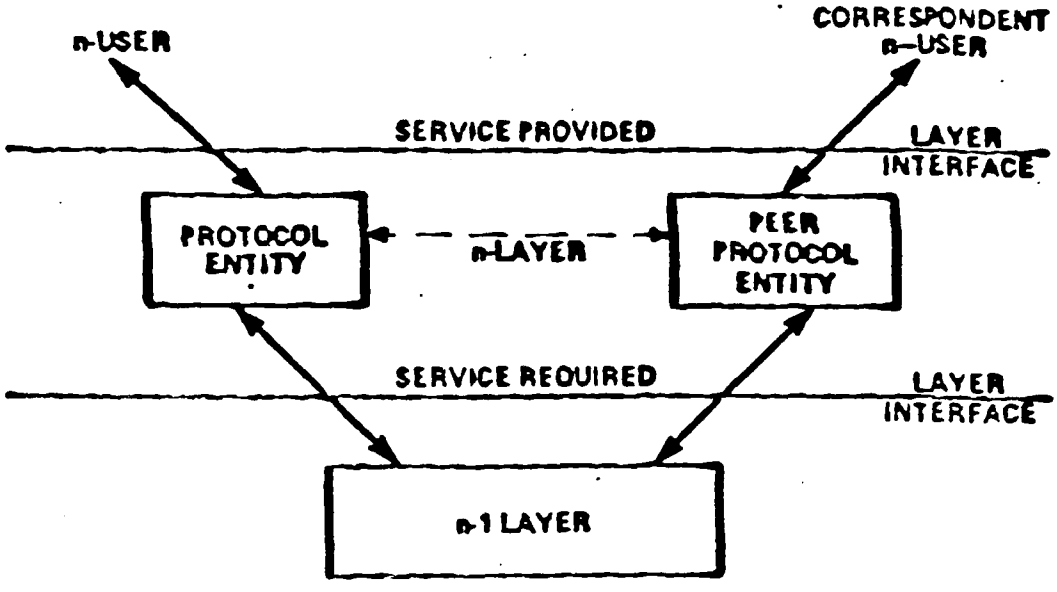


Fig. 1

TRADITIONAL DATA-PROCESSING		O. S. I.	
Operating Systems functions	Prog. Languages functions	Session Layer functions	Pres. Layer functions
RUN		Estab. Sess.	
Enqueue/Dequeue		Send/Receive	
Post / Wait		S/R Expedited	
Semaphore		Token	
	Co-routines	TWA dialogue	
	Procedure Call	(Part of TWA)	
	Data Types		Syntax
	Declarations		Syntax
			Negotiat.
	End	Release Sess.	
Check Points		Sync.	
Restart		Resync.	

Fig. 2

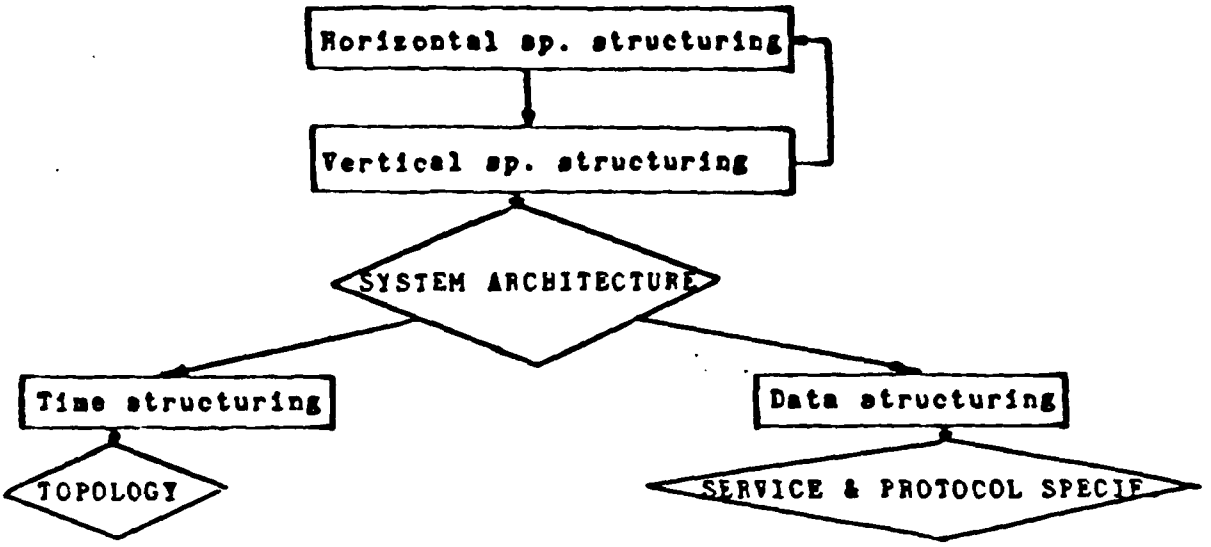

Horizontal sp. structuring
Vertical sp. structuring
SYSTEM ARCHITECTURE
Time structuring
Data structuring
TOPOLOGY
SERVICE & PROTOCOL SPECIF

Fig. 3

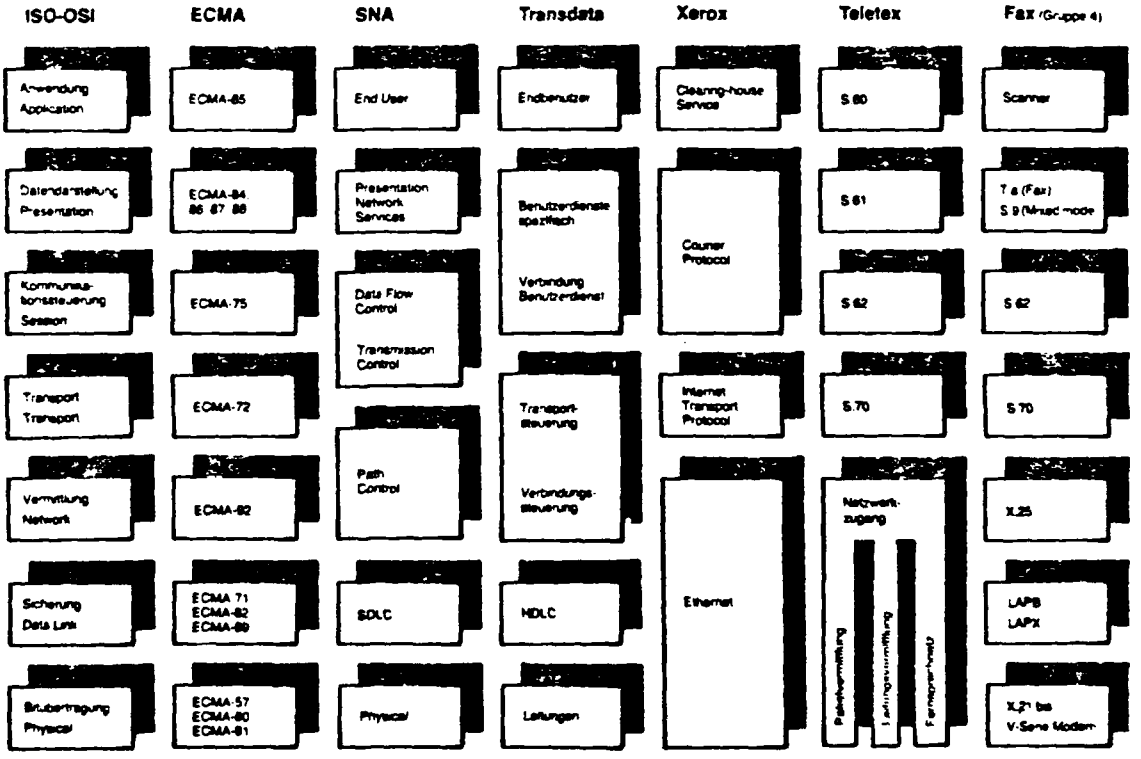

ISO-OSI
ECMA
SNA
Transdata
Xerox
Teletex
Fax (Gruppe 4)
Anwendung
Application
ECMA-85
End User
Endbenutzer
Clearing-house Service
S.60
Scanner
Datendarstellung
Presentation
ECMA-84, 86, 87, 88
Presentation Network Services
Benutzerdienste spezifisch
S.61
T.a (Fax)
S.9 (Mixed mode
Kommunikationssteuerung
Session
ECMA-75
Data Flow Control
Transmission Control
Verbindung Benutzerdienst
Courier Protocol
S.62
S.62
Transport
Transport
ECMA-72
Internet Transport Protocol
S.70
S.70
Vermittlung
Network
ECMA-82
Path Control
Transportsteuerung
Verbindungssteuerung
Netzwerkzugang
X.25
Sicherung
Data Link
ECMA-71
ECMA-82
ECMA-89
SDLC
HDLC
Ethernet
LAPB
LAPX
Bitübertragung
Physical
ECMA-57
ECMA-80
ECMA-81
Physical
Leitungen
X.21 bis
V.Serie Modem

Fig. 4

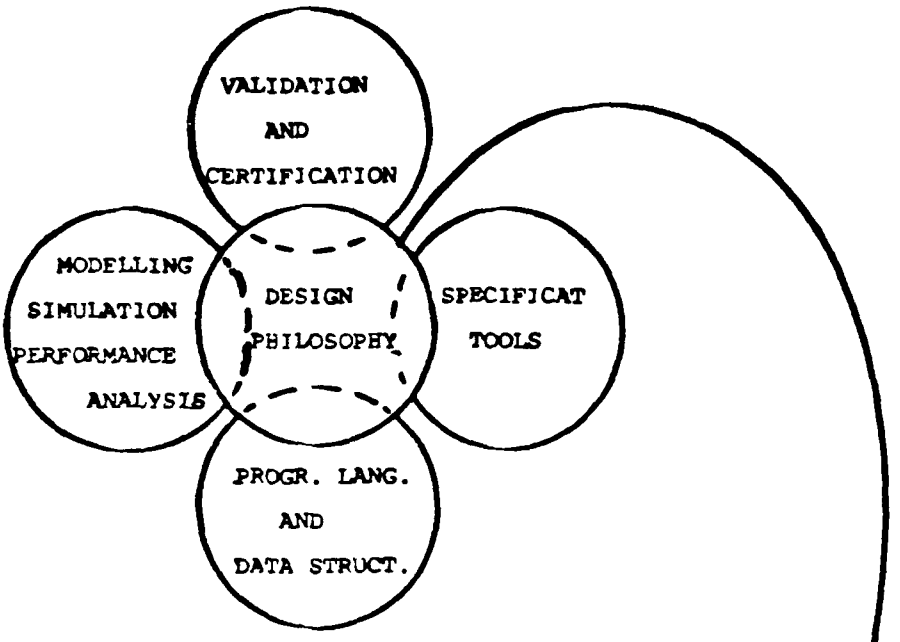

VALIDATION
AND
CERTIFICATION
MODELLING
SIMULATION
PERFORMANCE
ANALYSIS
DESIGN
PHILOSOPHY
SPECIFICAT
TOOLS
PROGR. LANG.
AND
DATA STRUCT.

Fig. 5

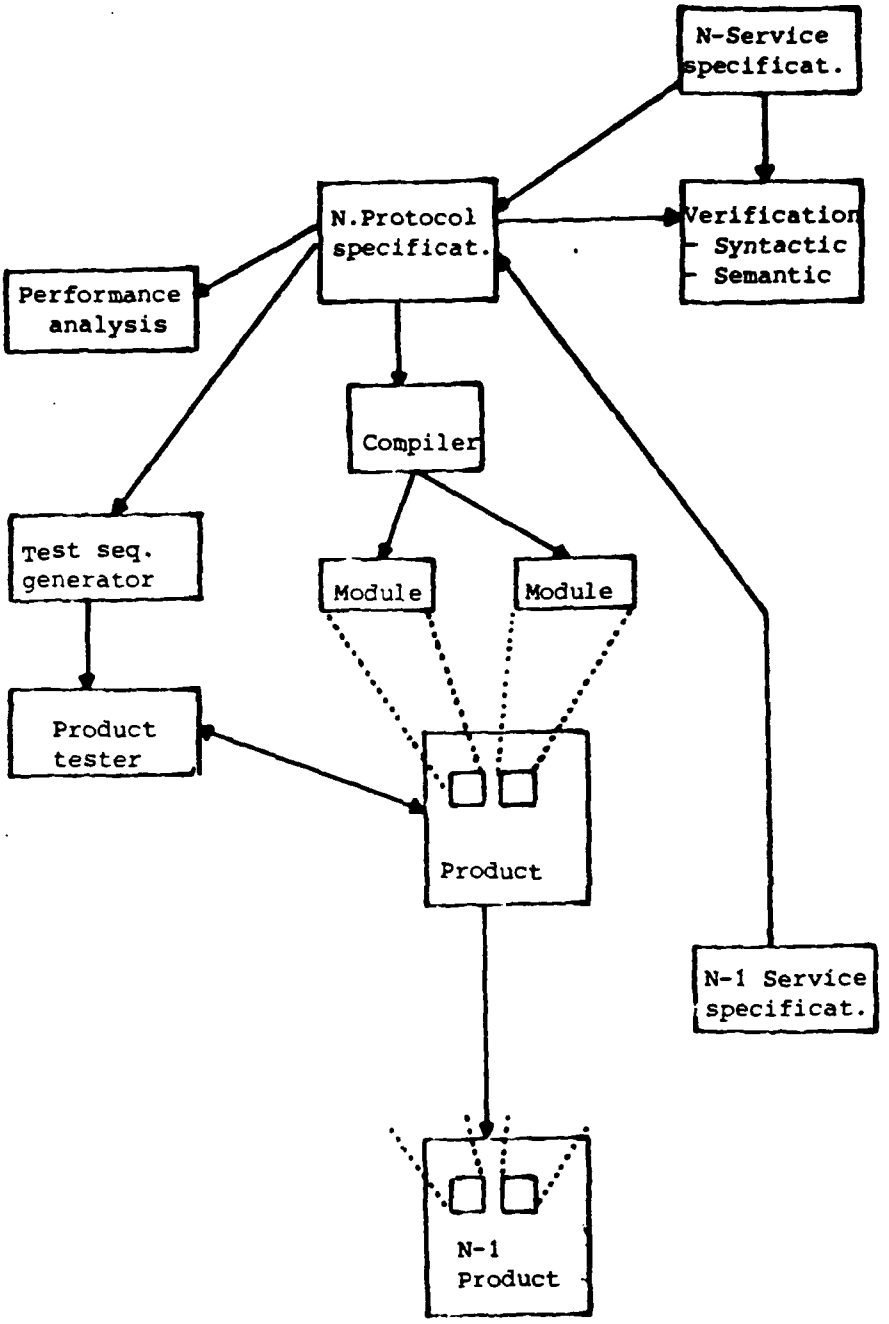

N-Service specificat.
N.Protocol specificat.
Verification
- Syntactic
- Semantic
Performance analysis
Compiler
Test seq. generator
Module
Module
Product tester
Product
N-1 Service specificat.
N-1 Product

Fig. 6

INDUSTRIAL LOCAL AREA NETWORKS

G. LE LANN
INRIA
PROJECT SCORE
B.P 105
78153 LE CHESNAY CEDEX
FRANCE

ABSTRACT

Such real-time applications as command-and-control or process control have been using computing systems for some time. Recently, with the advent of distributed computing systems, more attention has been paid to the real-time communications issue. Industrial local area networks are those sub-systems in charge of handling real-time communications. Requirements to be met by such sub-systems are presented. Recommendations for standards as currently proposed by the IEEE 802 Committee are discussed (the reader is supposed to be familiar with these recommendations). Finally, future trends and possible evolutions of industrial LANs are identified.

1. INTRODUCTION

In 1964, when Rand Corporation completed its D.O.D. report entitled "On Distributed Communications" [BAR 64], only a few people had an idea of how digital communication was to impact our world by the end of this century.

A few years after the DOD-Arpanet was started (1969) and became a tool used by thousands of people, public packet-switching data networks were installed in a number of countries.

These data networks are built out of existing analog telephone networks and are intended to offer reliable communication services across medium to large distances. More recently, the need for offering identical services over short distances has been widely recognized. The major driving force in this area has been Xerox which developed a local area network, called the Ethernet. In 1976, several versions of it were installed and used in a number of Xerox locations for the purpose of experimenting local area networking technology and assess its usefulness in the perspective of office automation.

Since then, and because of the importance of the office automation market, local area networks (LANs) have mushroomed.

Faced with anarchy, users and manufacturers felt it necessary to establish common rules and standards for this new market. In Europe, in the U.S.A., committees were put to work (ECMA, IEC, IEEE). In 1985, the IEEE 802 Committee has become the focus point for LAN standardization activities. Proposals approved within this Committee are forwarded to the International Standard Organization (ISO) before they can be adopted as international standards.

Bearing in mind that the IEEE 802 Committee concentrates on office-oriented LANs, one could consider that the 802 proposals are of no concern for real-time application oriented LANs, such as LANs installed in factories, plants, workshops, etc., which we will refer to as industrial LANs.

The main purpose of this paper is to discuss the most prevalent characteristics of 802 "standards" in the light of the communication requirements usually found in industrial environments as well as to identify some possible and/or desirable evolutions in this area.

2. COMMUNICATION REQUIREMENTS IN INDUSTRIAL APPLICATIONS

Industrial applications come under a large variety of different flavours. Automation in the industrial world is a continuous process, more functions becoming gradually automated and new functions being devised only because automation technology is there.

It is therefore a bit risky to state communication requirements without being very specific about both the type of application considered and the time at which such requirements are identified. Furthemore, it is not possible, in one paper, to describe in great detail all the various combinations of communication requirements. We will then take a simple approach and present those requirements most often encountered. We will leave to the reader the task of choosing which of these requirements apply to his/her particular application in the near-term future.

2.1. Robustness requirements

Robustness is to be taken as a combination of reliability and availability requirements. Both of these terms have received widely accepted definitions | RAN 78 |. Robustness can be achieved by the use of fault avoidance techniques (which result in the production of highly available modules) and by the use of fault tolerance techniques. It is admitted that it is only through the use of fault tolerance techniques that one can design and build a system (a communication system in our case) that achieves any arbitrary high degree of robustness.

We would like to stress the importance of robustness requirements and warn the reader to guard himself/herself against such false soothsayings as "the hardware will get more and more reliable", "errors are not all catastrophic", "exceptional situations can always be handled by human operators", etc... It is well known that what can happen does eventually happen. Situations thought to be "almost" impossible to occur as well as - and this is the worst aspect - situations that were not even predicted have the unpleasant property to show up one day or another. The more complex a system is, the more likely it is that "improbable" and faulty situations will appear.

Also, the "intrinsic" reliability properties of some hardware element do not mean too much, for the actual reliability depends greatly on the physical environment and on the level and type of noise.

In industrial applications, wrong computations (in the algorithmic sense) and wrong timing (in the chronological sense) are the enemy. Note that a valid computation, if performed too late, might result into a faulty behaviour of the system. This applies also to industrial LANs. It is then better for an industrial LAN not to deliver a message from time to time (because correct delivery was not possible) rather than to deliver incorrect messages. It is in general difficult or impossible to "compensate" the effects of a wrong output in a real-time environment. The consequence of this observation is that robustness issues and timing issues (see section below) cannot be addressed separately | MEY 80 |, although the requirements can be expressed independently.

Robustness requirements for industrial LANs can be derived from quantified objectives of many continuous/discrete process control systems, e.g. less than one fatal failure in five years or probability of a fatal failure less than 10^{-9} for ten consecutive hours. Clearly, LANs must be designed in such a way that they will assist in the repair process, by automatically providing outputs of internal tests they perform regularly. Repair interventions should not be needed for a LAN to continue to operate correctly. Fault detection and recovery or fault masking are definitely needed. They are the two facets of fault tolerance techniques which, as indicated above, are the only viable approach to the construction of robust LANs.

Fault tolerance is based on redundant hardware (e.g., physical links, physical communicating equipments), redundant software, (e.g., communication protocols, processes), redundant data (e.g., messages, system states). A small number of prototype or commercial LANs use some form of redundancy. Their robustness capacity is limited in the sense that it cannot be increased at will, so as to meet specific user requirements. What will be needed in the medium-term future are LAN architectures that are designed in such a way that they do not impose any artificial limitation with respect the degree of redundancy necessary to achieve a given degree of robustness.

The market for fault-tolerant systems in general is enormous and the fraction of it that will be captured by vendors by the mid-80's is estimated to be only in the order of 34 % | YAS 82 |. Robust industrial LANs have a bright future.

2.2. Timing requirements

An industrial LAN is the backbone of a distributed computing system that has to perform a number of tasks in "real-time", i.e. under specific timing constraints. For critical tasks, deadlines cannot be missed for this would constitute a system failure.

The tasks that must be performed by an industrial LAN are message passing tasks. Messages may be obtained from/sent by sensors, or sent to actuators, or communicated among processors and programmable automated devices. It is possible to identify different types of timing requirements. We will simply present three types of timing requirements, in increasing order of complexity.

In this presentation, occurence of faults and errors is not taken into consideration and a physical time reference, common to a LAN and its environment, is supposed to exist.

Among the various time variables of interest, let us concentrate on access delay, i.e. the time elapsed between submission of a message and its actual transmission on a LAN physical medium.

2.2.1. Probabilistic timing requirements

Such requirements are expressed as probability distribution functions, with no upper bound. Access delays are charaterized by an expected value, a variance and/or a confidence interval (e.g. 95% of access delays less than or equal to 600 ms). The environment is assumed to be able to tolerate (possibly largely) varying and theoretically unbounded transmission delays. Probabilistic requirements correspond to curve P in figure 1.

2.2.2. Deterministic timing requirements

Timing requirements are deterministic when the existence of an upper bound is guaranteed. In other words, there is always a predictable finite number of state transitions between message arrival and message departure, for any given message.

(i) Promptness requirements

Acces delays are characterized by an expected value, á variance and an upper bound. Such requirements correspond to curve DP in figure 1.

(ii) Timeliness requirements

In addition to promptness requirements, it might also be necessary to require that tasks are not run before physical time has reached some value. Thus the need for knowing a lower bound for access delays. Such requirements correspond to curve DT in figure 1.

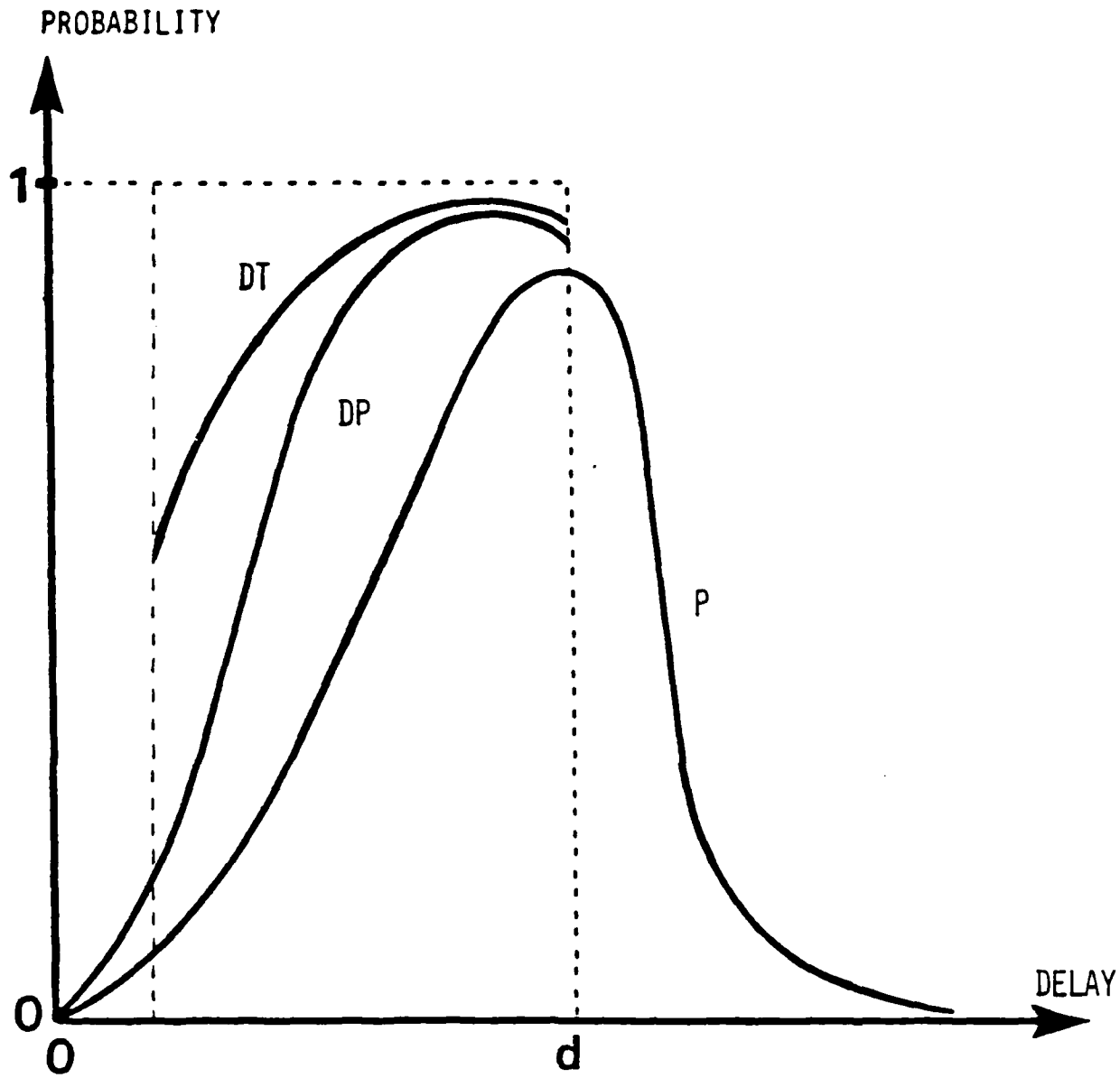


Figure 1 : Timing requirements

87

2.2.3. Comments

Timing issues are very controversial. Being at the root of problems difficult to tackle correctly, these issues tend to be ignored or treated very superficially. There are a number of myths used to convince users that timing issues are not a problem per se.

Some of the claims used by advocates of a probabilistic approach are as follows :

- measurements demonstrate that LANs are under-utilized (e.g. 15 % of available bandwidth) ; consequently, all messages are transmitted as desired

- all systems can fail and/or can become overloaded ; therefore, deterministic message handling cannot be guaranteed under all circumstances.

These claims are defeated by advocates of a deterministic approach as follows :

- measurements are only valid for the systems on which they are conducted, and they reflect some instantaneous utilization mode ; what about future evolution ? Who predicted traffic jams in 1925 ?

- reasoning in terms of average values, measured on time intervals which are orders of magnitude bigger than the sampling period of some external phenomenon to be controlled, is totally misleading and has no relevance at all in a real-time context. It is indeed the case that traffic peaks exist on small time slots and it is the case that such peaks must be absorbed in predictable time by a LAN.

- it is precisely when faults or overloads occur that the behaviour of a LAN must be predictable. Considering the occurence of faults as an excuse to overlook the timing issue would be equivalent to argue that because drunk drivers are the main cause of car accidents, it is then not necessary to equip cars with safe brakes.

2.3. Flexibility requirements

The notion of flexibility results from the recognition that physical systems in the large sense (mechanics, living beings, etc...) are affected by the passing of time. Human needs evolve, physical equipments break, new technologies are put to work and so on. All this above, when applied to LANs, leads to the notion of LANs that one should be able to modify at unpredictable times for the purpose of making them meet their existing specifications better or meet more sophisticated specifications.

The most important types of flexibility requirements are related to functionality (evolution of the services provided), implementation (integration of technological advances) and topology (evolution of the physical dimensions). Of course, it must be possible to perform these modifications without disrupting the functioning of an industrial LAN.

Intuitively, designing and structuring an industrial LAN with the aim of achieving all kinds of flexibility properties bears some similarities with designing and structuring with the aim of achieving fault-tolerance. Differences stem from the fact that "modifying voluntarily" a system allows for the execution of an explicit "separation" procedure before the actual modification takes place, which cannot be assumed to hold when faults occur.

Therefore, there are similarities for the structuring principles only. The types of algorithms needed in both cases are different.

We are currently witnessing the emergence of different industrial LANs, each of which being geared at various market niches, which correspond to different cost-effectiveness tradeoffs. We are also witnessing decreases in costs. The main factor behind this long awaited trend is the standardization work, which has made it possible for manufacturers (the VLSI circuits industry in particular) to embark upon the design and the fabrication of cheap sophisticated hardware that implements the low-level protocols agreed upon within the 802 Committee.

3. INDUSTRIAL LOCAL AREA NETWORKS IN PERSPECTIVE

As for every manufacturing activity, market trends are divided between standard compatible and non standard compatible products.

3.1.Proprietary industrial LANs

Either because they anticipated the need for industrial LANs before standards were developed or because they felt they could go along their own way or because they felt the standardization bodies would move too slowly, some manufacturers have promoted industrial LANs which do not meet 802 Committee recommendations. Examples are Allen Bradley's Data Highway, Gould Modicon's Modbus, Texas' TI Way I and Intel's Bitbus.

Some LANs are selling well while some others have failed to penetrate the market (e.g. Modicon's Modway). Whether a proprietary approach is likely to succeed depends on many parameters, among which one finds financial health of the parent company and good engineering of the products. Selection of a proprietary industrial LAN entails specific medium-to-long term commitments that may not be obvious at first glance. For instance, SDLC, the link protocol used in Bitbus, is incompatible with 802.2 (link) specifications. The fact that IBM owns some 20% of Intel shares could suggest that such a choice is not a mere accident.

3.2."Standard-compatible" industrial LANs

Imagine a poll is conducted about the following question : "Costs not being accounted for, which is the IEEE 802 recommendation which looks most applicable to industrial LANs ?". Likely, the results would be 802.4 (token bus) ranked first, 802.3 (contention bus) second and 802.5 (token ring) third. Why ?

LANs which provide "deterministic" services (802.4 and 802.5) are favored against those providing "probabilistic" services only (802.3). However, token rings have some drawbacks in industrial environments. A token ring relies on an active topology (active ring access units). It is inherently less robust than a passive bus. A large number of industrial LANs span short distances (e.g. less than 1 kilometer). The complexity of a token ring does not seem necessary for such LANs. A passive bus (802.3 in particular) is simpler to manage . Stars, rooted/unrooted trees and meshed topologies are most familiar in an industrial environment. These topologies are bus oriented, not ring oriented. Control of time intervals spent in transmitting messages is more accurate with token busses than with token rings . In particular, starvation is less likely to occur when using the four timers available with 802.4 busses than when using the unique timer and the static priorities available with 802.5 rings.

Finally, busses (mainly contention busses) have been put into operation in thousands of locations. This is not the case yet with token rings, whose IEEE 802 approval comes after official approval of 802.3 proposal (1983) and 802.4 proposal (1984).

Imagine now that costs are taken into consideration (which is usually the case in the real world). Depending on how the various requirements (see section 2) are weighted, one could have either 802.3 busses or 802.4 busses ranked first. The main reason why contention busses could win against token busses is the availability of several silicon versions of 802.3 protocols, which can currently be fully implemented with no more than two VLSI circuits. The production of 802.4 and 802.5 VLSI circuits lags behind. This is due to early IEEE 802 approval of a contention bus proposal and also to the relative simplicity of 802.3 protocols compared to 802.4 and 802.5 protocols. As costs always are an important practical issue, a large number of users might be prepared to sacrifice costly "determinism" and to adopt cheap "probabilism". This will certainly be the case when the timing requirements are not too stringent and/or when the devices to be connected are cheap. Of course, cheap determinism would be ideal ! But, after all, is it only a dream or could it be real ?

3.3. "Deterministic" versus "probabilistic" services

It is somewhat puzzling to observe that false statements, or at least overly simplified statements, keep being propagated and trusted. This is exactly the case with the controversy concerning token-passing LANs (802.4 and 802.5), which are labelled as "deterministic" LANs, and contention LANs (802.3), which are labelled as "probabilistic" LANs. To begin with, it might be useful to remember the exact meaning or determinism, which is existence of a finite number of system state transitions for switching from one state (e.g. message submission) to another state (e.g. successful message transmission). Determinism is a logical concept. In other words, it is not because an algorithm is deterministic that a LAN making use of such an algorithm can always meet given timing constraints. Many parameters have to be taken into account in order to compute the exact physical values of expected upper bounds. If these values are too high, determinism does not help at all.

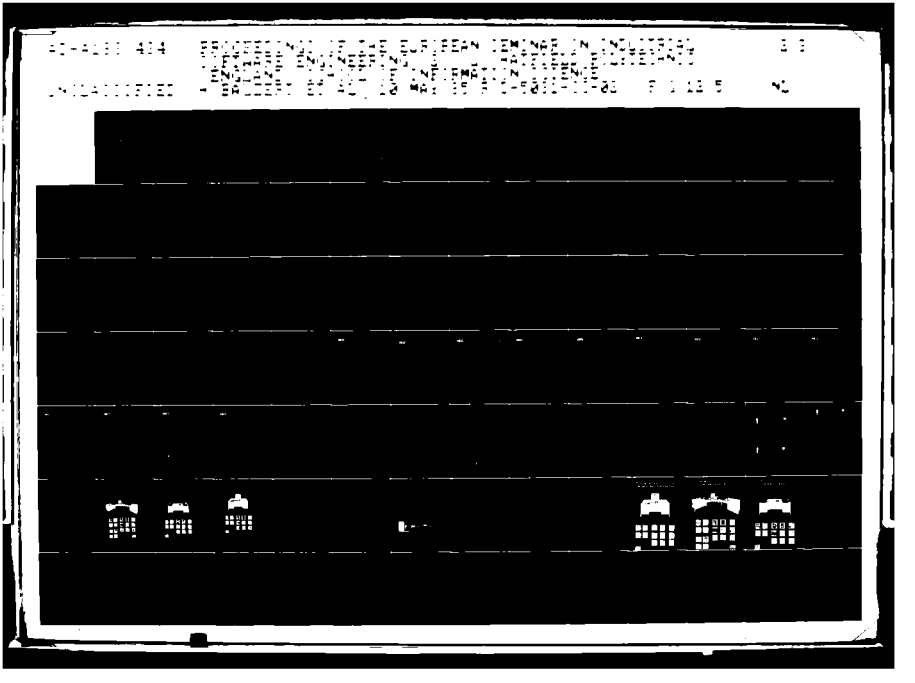
3.3.1. Token passing LANs are "deterministic"

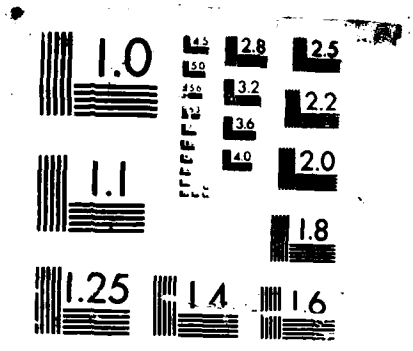
Why is it so that 802.4 and 802.5 LANs are considered as being "deterministic" ? Let us concentrate on those messages which are first in the waiting queues of the various access units. Let us consider token rings first. Can a token-ring guarantee that each of these messages will be transmitted in bounded finite time ? The answer is yes if static priorities are not used (but what about those units which handle very critical and urgent messages ?). The answer is definitively no if static priorities are used. It is well known (see queuing theory) that when static priorities are used, only the clients with the highest priority enjoy guaranteed service. For all other clients, starvation can occur. In other words, some messages might be denied access to a ring for ever. Is this a deterministic service ?

Let us consider token busses now. Access units make use of four timers, one of them (class 6) corresponding to a guaranteed time interval used to transmit most urgent messages. A burning question is how to compute the "good" values of these timers so as to keep token rotation time to a "reasonable" value. These computations must integrate some straightforward variables (e.g. maximum number of access units, maximum physical length of the bus, etc.) but also more subtle variables. For example, one must know how often every unit will decide to "leave" the logical ring, how often a unit must "sollicit" missing units which are not on the logical ring but which would like to join in, how long is the "insertion" procedure execution when many units collide in response to a "sollicit" frame, etc. These variables depend on assumptions made on the nature of the input traffic. Bad news ! Except in very specific cases (which do not represent the vast majority of potential 802.4 bus users), input traffic assumptions are of a probabilistic nature and so is the percentage of time spent in executing the leave/insert logical ring protocols. Therefore, one has to admit that 802.4 busses upper bounds are probabilistic.

Let us look now at the fault handling issue. With token passing LANs, it is more difficult to predict what impact faults might have on access delays than with contention LANs. Such faults as unit crashes or electromagnetic noise do not impact 802.3 LANs very much because no global variable must be protected against these faults, to the exeption of physical signals. Conversely, not only such physical signals must be guarded against faults with 802.4 or 802.5 LANs, but also the token variable (MAC level), which in the case of token rings carries also the vital priority indicators. The recognition of the need to recover from token losses led to the conclusion that a single unit should be designated as the control (central) unit. Should this unit fail, another one is elected as the new control unit. Now the questions : "how does one know how often a token is lost ? ", "how often does a control unit go down ?". Will the answers provide deterministic information or will we be playing with probabilities ?

There is another more subtle point which is that 802.4 and 802.5 [illegible] which have been designed from the start to eliminate [illegible] eliminate collisions at all when election of a new control unit [illegible] when 802.4 "sollicit" frames are transmitted). Unfortunately [illegible] how such collisions can be resolved deterministically [illegible] time. We leave it to the reader the task of drawing [illegible] discussion above.





MICROCOPY RESOLUTION TEST CHART

3.3.2. Contention LANs are "probabilistic"

3.3.2.1. CSMA-CD compatible LANs

It might be the case that because CSMA-CD protocols belong to the family of random acces protocols, CSMA-CD protocols are regarded as behaving probabilistically ! One might also be tempted to believe that there is only one way to resolve collisions, that is the Ethernet way. Although it is irrefutable that the Binary Exponential Backoff algorithm is of a probabilistic nature, it is wrong to state that contention LANs must be probabilistic in general. Space of choices is given in figure 2. As can be seen, it is not because initial accesses can lead to collisions that the situation is hopeless. A large number of algorithms which provide contention LANs with deterministic behaviour have been published in the literature.

See, for example, | CHL 79 |, | CHL 80 |, | FRA 80 |, | KUR 83 |, | MAS 81 |, | MOL 81 |,| POW 81 |, | ROM 81 |, | TOB 80 |, | TOB 82 |, | TOW 82 |.

Among these numerous proposals, it suffices to choose those which are 802.3 compatible to obtain a "standard" CSMA-CD LAN which is at least as "deterministic" as token passing LANs. The interest in "deterministic" contention LANs is so high that prototypes have been or are being built in Europe (France, Germany and Netherlands at least), in Israel, in the U.S.A. and in Japan. The potential commercial success of this approach lies in the low prices reached by CSMA-CD access units. If the physical "intervention" needed to implement a deterministic collision resolution scheme is limited, in complexity and in cost, then deterministic 802.3-like LANs could fly soon. Being deterministic , such LANs could guarantee that all messages involved in a collision are transmitted in some bounded finite time. Therefore, such LANs could be used to carry all kinds of traffic mixes such as aperiodic data packets and periodic voice packets.

3.3.2.2. High - speed LANs

Deterministic CSMA-CD can only be used when CSMA-CD achieves efficient channel utilization, i.e. when the ratio of the propagation delay over message duration is small compared to 1 (less than 0.2 is good practice). When not the case, i.e. for large LANs (metropolitan area networks) or for high-speed LANs, neither CSMA-CD nor explicit token-passing protocols are appropriate. The overhead incurred for every token passing operation (token handling protocol execution), for token transmission on the medium and for lost token recovery, is fixed and largely independent of the bandwidth available. Largely independent of the bandwidth as well is the time wasted in reconfigurating physical/logical rings. Acceptable at "low" bandwidth (lower than a few dozens of Mbits/s as an indication), this overhead becomes unbearable at higher speeds, such as those attained with optical transmissions.

Being forced to throw away CSMA-CD and explicit token passing protocols for high-speed LANs , it looks like the only choice left is the well known family of synchronous time division multiplexing protocols. Not quite so. A fair amount of

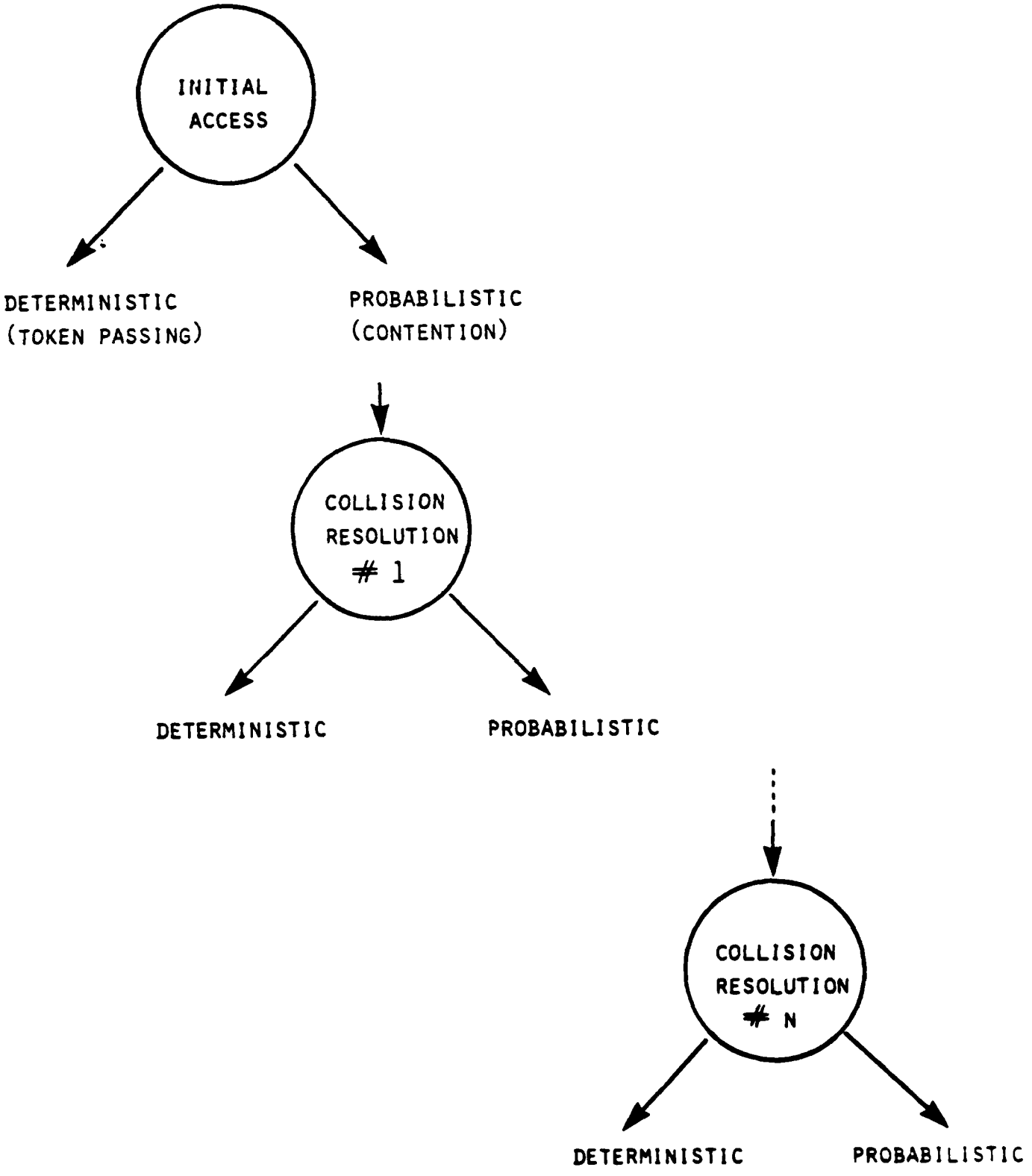


Figure 2 : decision tree for collision handling

work has been invested in the identification of contention protocols that would achieve a very good utilization ratio of high bandwidth channels. In an exellent survey paper, most of these protocols are presented and evaluated against each other | FIN 84 |.

3.4. Higher-level protocols

Apart from the question of which is the best 802 MAC protocol, it is necessary to examine which types of higher-level protocols come with current industrial LANs. There is a trend toward the integration of all protocol layers ranging from 1 (physical) to 5 (session) on a single board, used as an attachment unit to a LAN. But only a few manufacturers actually deliver such boards presently.

A global observation can be made. Transport and session protocols (when available) which are implemented on industrial LANs provide more services than specified in the ISO/Open System Interconnection Reference Model. In particular, it is often the case that broadcast and multicast datagram services are made available at layers 4 or 5.

Conversely, there is no more "real-time" ingredient coming along with industrial LANs protocols than with conventional LANs or WANs protocols. Designers and users of industrial LANs might realize soon that they have to depart from traditional ISO-like high-level protocols if they want their networks to retain the benefits of deterministic multi-access schemes.

For instance, such services as guaranteed delivery of datagrams and dynamic priority-based scheduling may be needed in industrial environments. More general types of interprocess conversations than just connectionless and connection-oriented point-to-point communications may also be desirable. Such issues as maintaining "real-time" services across bridges and gateways for interconnected industrial LANs must be addressed thoroughly.

4. CONCLUSION

Is there a conclusion ? With respect to principles, to the algorithmic nature and the properties of the various protocols examined, conclusions can be established. This has been done throughout the paper.

With respect to current and future trends of the industrial LAN market, it is difficult to conclude and to make predictions. Initiatives undertaken by large manufacturers are, by definition, unforseeable. We have witnessed such an initiative in 1984 with the announcement of M.A.P. by General Motors. The impact this announcement has had is very much comparable to the impact produced by IBM announcements in other areas. In 1982, IBM said very clearly that the distribution of documents within the 802.5 project should not be interpreted as a pre-announcement of a new product. The result has been that all users have been expecting such an announcement since then. Again, in 1984, IBM re-stated its intention to deliver its first token rings no sooner than 1986. However, in 1984, a few manufacturers have proudly announced LANs which are IBM token ring compatible !

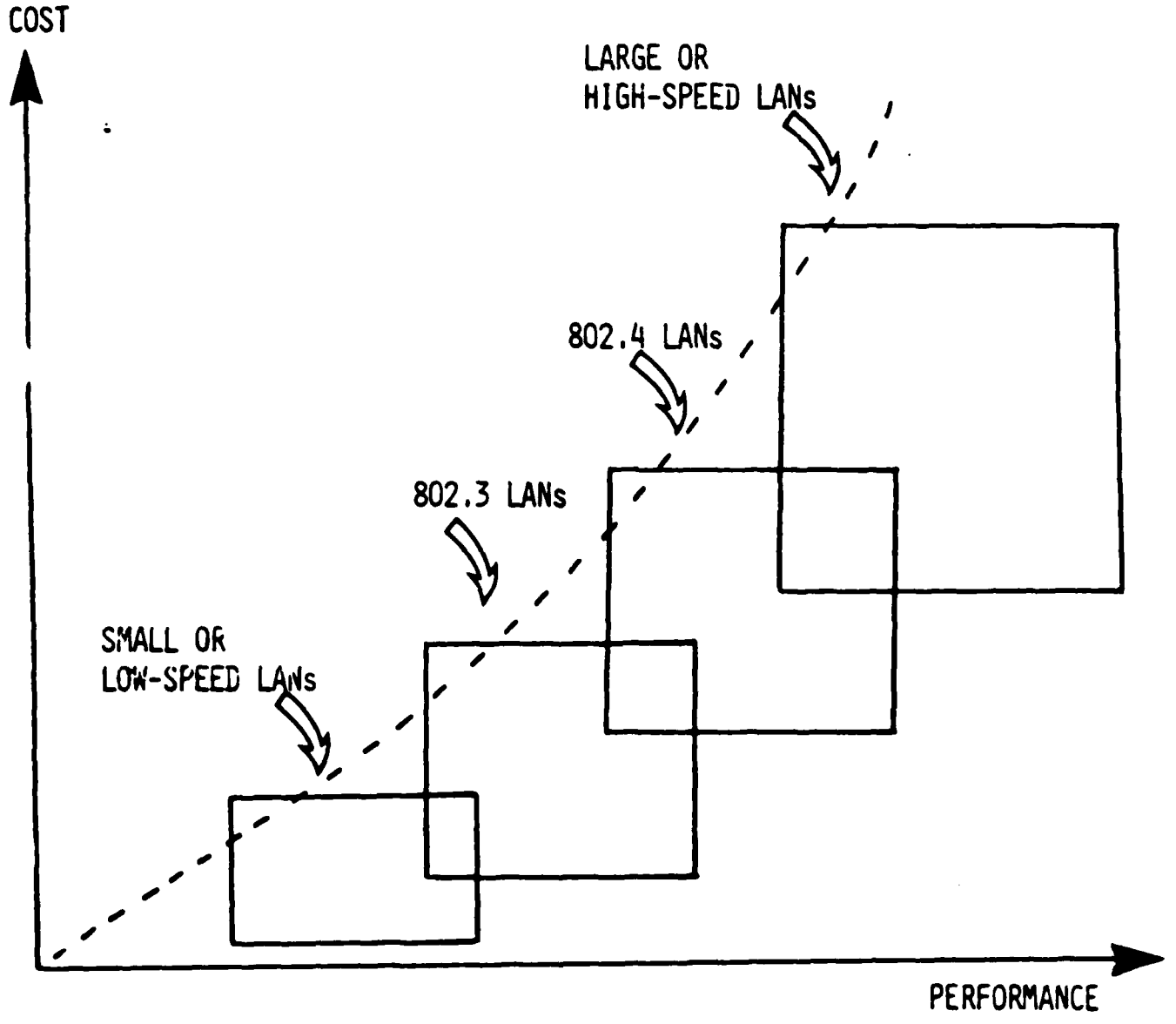


Figure 3 : a simplified segmentation of the industrial LANs market

A similar psychodrama is developing with MAP. At this time of writing, not all MAP layers are specified. The target date for a complete specification is 1988. Nevertheless, some manufacturers are currently preparing themselves to announce LAN oriented products which are fully MAP compatible !

If we step back a little, we can see that there is room for everybody. Four major segments of the industrial LANs market can be identified, as indicated in figure 3. They correspond to different cost-effectiveness ratios. As can be inferred, one important market niche corresponds to gateways and bridges, which are needed to allow the various LANs to be found in real-time environments, whether "IEEE 802 - compatible" or not, to talk to each other. The potentially brilliant future of 802.4 busses as industrial LANs, thanks to General Motors support, could be challenged in two different ways. A "low-end" challenge could develop with the possible advent of deterministic baseband/broadband CSMA-CD LANs. A "high-end" challenge could develop with the future advent of affordable optical transmission technology. Only time will tell.

REFERENCES

| BAR64 | P. Baran, "On distributed communications", Rand Corporation Series Reports, Santa Monica, August 1964, 453 p.

| CHL79 | I. Chlamtac, W.R. Franta, K.D. Levin, "BRAM : the broadcast recognizing access method", IEEE Transactions on Communications, Com-27, n°8, August 1979, 1183-1190.

| CHL80 | I. Chlamtac, W.R. Franta, "Message based priority access to local networks", Computer Communication, Vol. 3, 2, April 1980, 77-84.

| FIN84 | M. Fine, F.A. Tobagi, "Demand assignment multiple access schemes in broadcast bus local area networks", IEEE Transactions on Computers, vol. C-33, n°12, December 1984, 1130-1159.

| FRA80 | W.R. Franta, M. Bilodeau, "Analysis of a prioritized CSMA protocol based on staggered delays", Acta Informatica, Vol. 13, Fasc. 4, 1980, 299-324.

| KUR83 | J.F. Kurose et al., "Controlling window protocols for time-constrained communication in a multiple access environment", ACM Sigcomm, Vol. 13, 4, October 1983, 75-84.

| MAS81 | J.L. Massey, "Collision-resolution algorithms and random-access communications", in Multi-User Communication Systems (Ed. G. Longo), Springer-Verlag CISM n° 265, 1981, 73-137.

| MEY80 | J.F. Meyer, "On evaluating the performability of degradable computing systems", IEEE Transactions on Computers, vol. C-22, August 1980, 720-731.

| MOL81 | M.L. Molle, "Unifications and extensions of the multiple access communications problem", UCLA report n° CSD-810730, July 1981, 131 p.

| POW81 | D.R. Powell, "Réseaux locaux de commande-contrôle sûrs de fonctionnement", Thèse d'Etat, INPT, October 1981, 206 p.

| RAN78 | B. Randell, P.A. Lee, P.C. Treleaven, "Reliability issues in computing system design", ACM Computing Surveys, vol. 10, 2, June 1978, 123-165.

| ROM81 | R. Rom, F.A. Tobagi, "Message-based priority functions in local multi-access communication systems", Computer Networks, 1981, 273-286.

| TOB80 | F.A. Tobagi, R. Rom, "Efficient round-robin and priority schemes for unidirectionnal broadcast systems", in Local Networks for Computer Communications (Eds. A. West, P. Janson), North-Holland/IFIP, 1980, 125-138.

| TOB82 | F.A. Tobagi, "Carrier-sense multiple access with message-based priority functions", IEEE Transactions on Communications, Com-30, January 1982, 185-200.

| TOW82 | D. Towsley, G. Venkatesh, "Window random access protocols for local area networks", IEEE Transactions on Computers, C-31, 8, August 1982, 715-722.

| YAS82 | E.K. Yasaki, "Fail-safe vendors emerge", Datamation, November 1982, 51-58.

JOHN FAVARO

Unix* - A VIABLE STANDARD FOR SOFTWARE ENGINEERING?

Introduction

The catchword "standard" has been used in conjunction with the Unix operating system with increasing frequency in recent times. This situation reflects a growing need among software developers for standards not merely at the programming language level, but at the level of the programming environment.

Evidently, Unix has been perceived as the current best hope of achieving that goal. Yet there are significant obstacles to the standardization of Unix. In this paper we will take a look at recent efforts toward the standardization of Unix and examine some of the problem areas in detail.

Versions of Unix

Before discussing the standardization of Unix, we should first consider the motivation that led to the standardization effort: namely, the many versions of Unix.

---

* Unix is a trademark of AT&T Bell Laboratories

Currently, Unix systems fall into three basic categories:

- those systems being marketed by AT&T itself.
- those systems that are Unix-based, but marketed by other companies with a license from AT&T.
- "Look-alikes": those systems that are marketed *without* a license from AT&T.

AT&T alone markets a number of versions of Unix, including Version 6, PWB, Version 7, 32V, System III and System V.

The bulk of the other Unix versions fall into the second category. They tend to be based upon Version 7 or System III. Systems in this category include XENIX, UNIPLUS, VENIX, IS/3, UTS.

It is into this second category that Berkeley Unix falls. Berkeley Unix was originally based upon 32V, and is now being distributed as "Berkeley Software Distribution 4.2".

Some examples of systems in the third category include COHERENT, IDRIS and UNOS.

This adds up to a staggering number of versions of Unix indeed! Yet this picture is somewhat deceiving; for, only four of these versions have really been taken into serious consideration in the standardization efforts:

Version 7

Version 7 marks the beginning of "modern times" for the Unix system. Up to that point, the system had been mainly used in research circles, and had not yet acquired the facilities now considered to be basic to the Unix programming environment, such as the standard I/O library.

Although Version 7 was officially released in 1979, it already existed as early as 1977 when Unix 32V was derived from it. It was the first port of Unix to the VAX, and laid the foundation for Berkeley Unix.

In fact, Version 7 owes much of its significance to the fact that it was the first version to be ported in earnest to the micros. In 1980, ONYX made the first micro port of Version 7, to the Z8000. Others followed suit quickly. In particular, Microsoft ported XENIX to a number of micros.

Version 7 was the last version of Unix that was actually produced by the research group that originally developed Unix. For that reason, the name was changed in later commercial releases.

System III

AT&T released System III in 1981. This was AT&T's first attempt to support Unix officially. A new pricing policy

was introduced with System III, which finally provided for binary sublicenses in place of the exhorbitantly expensive source licenses that had previously been required. Technically, System III consolidated the best of Version 6, PWB and Version 7.

But the real significance of System III was the commitment of AT&T. This provided the necessary confidence needed by commercial vendors to base their derivatives on System III, and as a result there are quite a few commercial systems now based on System III.

System V

With the introduction of System V in January 1983, AT&T consolidated its Unix marketing effort. Up until this point, System V had actually been in use *internally* at Bell Labs, but System III was being marketed *externally*. Now this discrepancy no longer existed.

Commercial support was strengthened even further, with the introduction periodic updates and hotlines to support centers.

Recently System V has been upgraded with virtual memory and file locking facilities.

Berkeley 4.2 BSD

In 1976, Ken Thompson spent a year at the University of California at Berkeley, bringing Unix with him. At this time, a period set in of enhancements so important that is no longer possible to leave Berkeley out of a thorough discussion of Unix.

These enhancements include:

- The C-Shell. The most important alternative to the Bourne Shell, the C-Shell is preferred by many for interactive use because of facilities for aliasing (renaming commands) and command history lists for recall execution.
- Improved terminal handling. The *curses* and *termcap* packages, as well as the screen editor *vi*.

In 1979, with the 3.0 Berkeley Software Distribution (BSD), virtual memory came to Unix. The large address space paved the way for new applications - for example VAXIMA, the Berkeley implementation of the MACSYMA symbolic and algebraic manipulation system originally developed at MIT.

With the release of 4.2 BSD in October 1983, networking came to Berkeley Unix. Communication facilities based upon the U.S. Department of Defense standard Internet protocols

TCP/IP were integrated into the system. Furthermore, the file system was redesigned for higher throughput by taking advantage of larger block sizes and physical disk characteristics. These two additions alone were sufficient to insure Berkeley Unix an important position in the Unix world today.

It is these four Unix variants upon which we will focus our attention in the following discussion; for, taken together, they raise all of the major issues of the current standardization effort.

The Formation of /usr/group

The standardization problem actually began as early as 1979, when the first ports of Version 7 were undertaken. In seemingly no time at all, many variants sprang up in the commercial world. In recognition of this development, the /usr/group organization was founded soon afterwards in 1980. The organization took its membership from the commercial world, yet was vendor-independent.

It did not take long for the members of the organization to come to a decision about how they wished to spend their time: only a year later, in 1981, the /usr/group Standards Committee was formed.

The Standards Committee included participants from a broad range of vendors. Conspicuously, one of the most enthusiastic and active participants was AT&T. The committee set as its goal a vendor-independent standard for commercial applications.

The 1984 /usr/group Proposed Standard

The result was presented in 1984 in the form of a proposed standard[1]. The long term goal set for this proposed standard is ANSI and ISO Standardization, and indeed, an IEEE "Working Group on Unix Operating System Standards" has since been formed to pursue this goal, using the proposed standard as a base document.

What is contained in the proposed standard? Actually, this question is best approached by asking its complement: What has been *left out* of the proposed standard?

The proposed standard does not specify:

- The user interface (shells, menus, etc.)
- user accounting
- terminal I/O control
- in fact, *most* of the over 200 commands and utilities of the Unix system!

Then, we might ask, what *is* in the proposed standard? The standards committee decided that the best way to

achieve portability would be to concentrate on only two areas:

1. *System Calls*

The Unix kernel interfaces with applications programs through a set of system calls. These shield the programs from such internal matters as details of memory management, scheduling, I/O drivers, etc. These calls are described in Chapter 2 of the Unix *Programmer's Reference Manual*. The proposed standard defines a set of 39 system calls that are to be used by all applications programs.

2. *The C Language Libraries*

Chapter 3 of the Unix *Programmer's Reference Manual* describes the set of library routines normally available to programs written in the C language. The proposed standard defines a version of this library that is to be used by applications programs.

File Locking

In the entire set of system calls specified in the proposed standard, only one *extension* to the set of system calls available on most Unix systems (in 1984, at least) appeared: file and record locking.

This fact testifies to the enormous importance of file and record locking in commercial and data base applications.

The new system call is *lockf(2)*. It may be used for record locking and for semaphores. Using *lockf*, a program may define "regions" of a file which are locked individually.

lockf may be used in either an "advisory" or "enforced" form. The advisory form may be circumvented by using normal *read* or *write* system calls. Thus, the advisory form assumes that the processes are explicitly cooperating with each other in a "friendly" way.

The enforced form protects a *region* even from those who have no knowledge of the facility. The *lockf* definition specifies that deadlock must be guaranteed between processes related by locked resources.

As defined, *lockf* represents a compromise: on the one hand, processes not using locks don't need to know about them; but because of this, deadlocks may occur.

The System V Interface Definition

We noted earlier that AT&T has been an active participant in the /usr/group standardization effort. Thus it is not surprising that the proposed standard basically reflects the AT&T world.

It would be quite a feather in AT&T's cap to have an official Unix system standard whose contents correspond to the flavor of Unix marketed by AT&T. The /usr/group proposed standard represents the first step in that direction.

The second step in that direction was taken in January of 1985 with the announcement of the System V Interface Definition[2]. This document defines a minimum set of system calls and library routines that should be common to all operating systems based on System V. If that sounds familiar, it is no coincidence: the document is virtually identical to the /usr/group proposed standard in content. A separate chapter of the document carefully describes those areas in which differences remain, and includes plans for eventual migration towards total compatibility.

AT&T is attempting to back up this document with promises of adherence of future releases of System V to the Interface Definition. These promises have taken the form of two so-called "Levels of Commitment". Each component in the interface has a commitment level associated with it. A component with Level 1 will remain in the definition and be migrated in an upwardly compatible way. A component with Level 2 will remain in the definition for at least three years (although it could be dropped later).

The /usr/group proposed standard defines, as discussed before, only a minimum set of functions and ignores the many commands and utilities of Unix. But AT&T was interested in capturing the full functionality of System V in its definition. The solution adopted was to unbundle System V into a "base" and "extensions". The "base" part corresponds to the /usr/group standard.

The components of the base fall into the following categories:

- Operating system services
- error conditions
- Signals
- Other library routines
- Header files
- Utilities
- Environmental variables
- Data files
- Directory tree structure
- Special device files

Even within these categories, the components are more finely partitioned. For instance, within the Basic Operating System Services category, the low-level process creation primitives fork and exec are segregated into a group that should be avoided whenever possible in favor of the more general system primitive. Similarly, the use of

low-level *read* and *write* operations is discouraged whenever routines from the Standard I/O Library will suffice.

A System V Kernel Extension has also been defined. The functions provided in this set have mostly to do with the semaphores and shared memory of System V. Why wasn't this directly included in the base? We will have more to say about the problems of compatibility at this level later.

The other planned extensions fall into these categories:

- Basic Utilities
- Advanced Utilities
- Software Development
- Network Services
- Large Machine
- Graphics
- Basic Text Processing
- User Interface Services
- Data Base Manager

Verification Test Software

To tie all of this together, AT&T has reached an agreement with Unisoft Systems for the development of a verification test software package. This test software will determine whether a derivative of System V actually meets the

definition. Clearly, AT&T hopes that this validation suite will attain *de facto* the same kind of status in the Unix world that, say, the Ada Validation Suite has in the Ada world for conferring the official stamp of approval on a system.

Fundamental Obstacles to Standardization

When we consider the two areas on which the proposed standard concentrates, we find that one of them is relatively unproblematic: the C Language Libraries. The libraries contain a set of stable, well-understood routines, which can be ported to different Unix variants with little trouble. Even the C language itself is about to achieve standardization -- the ANSI X3J11 group is on the verge of defining a standard for C, based upon System III. Thus it will not be a radical departure from the current situation, but rather a codification of the language as it is today.

It is the other area addressed by the proposed standard that presents the major obstacles to a successful standardization attempt: namely, the system calls. For, the system calls reflect to a much greater extent than the C libraries the basic structure of a Unix variant's kernel.

Currently, several distinct groups rely on the defferent set of system calls available on the major versions:

- Such *de facto* industry standards as XENIX have heretofore been based on the Version 7 kernel. In recognition of this, the proposed standard "strives for compatibility with Version 7 whenever possible".

- The proposed standard has based its set of system calls on System III.

- The recently defined *Portable Common Tool Environment*[3] owes much of the nature of its system calls to System V. In addition, the proposed standard will eventually migrate toward System V.

- Finally, the majority of the participants in the world of high-performance networking workstations relies heavily on the kernel facilities provided by Berkeley Unix.

Where do the incompatibilities lie that give rise to these different groups? Although incompatibilities exist in many places, such as differing file system implementations, the problem can best be characterized by the differences in one particular area: *interprocess communication*.

Let us take a closer look now at the facilities provided by each of the four major versions for interprocess communication.

Interprocess Communication - Version 7

The following basic set of system calls related to interprocess communication was defined in Version 7:

signal: This defines the response to *pre-defined* external events, such as interrupts, alarms, hangups and hardware errors.

kill: This call is used to send signals to processes.

pause: A process suspends its execution pending receipt of a signal.

wait: A process waits for the termination of a child process.

pipe: The well-known interprocess byte stream.

Now, these facilities are adequate for most time-sharing applications, but not for such applications as networking and data bases. Where are the deficiencies? Basically, the problem here is one of flexibity. Among others, the following cases of inflexibility can be identified:

signal only allows pre-defined signal types. Therefore, no extra information can be conveyed.

kill can only send to single processes or to all processes.

pipes can only be used by *related* processes.

In System III, facilities were introduced to address exactly such problems of inflexibility.

Interprocess Communication - System III

Three new facilities were added in System III to handle the problems discussed above:

First, the so-called "FIFO" file was introduced. The FIFO file is a special file, just like a pipe. But FIFO files have unix system file names, not just file descriptors (thus they are also called "named pipes"). Furthermore, and more importantly, they can be used by *unrelated processes*, thereby increasing their flexibility enormously.

Secondly, two new, *user-defined* signals were introduced into the set of allowable signals. This allowed extra application-specific information to be conveyed with signals.

Thirdly, *process groups* were introduced. Thus a common basis was established for sending signals. Now the *kill* system call could send signals to all members of a process group.

Yet even these improvements had their problems. Two specific ones may be identified:

- Signals are not a sound basis for interprocess

communication and synchronization. They are not queued, so signals could be lost. The /usr/group proposed standard explicitly discourages the use of signals for synchronization.

- Pipes, even named ones, are inadequate in crucial ways. For instance, the byte stream of the pipe is simply too low-level. Message boundaries are not preserved, which in many applications can be a problem.

Additionally, pipes are very slow. To see why, consider the flow of data in a pipe: Data starts in the sender's address space. It is then copied into a kernel buffer. Finally, it is copied *again*, from the kernel buffer into the receiver's address space.

In short, the System III facilities cannot be considered to be a sound basis for process synchronization. With System V, such a basis appeared for the first time.

Interprocess Communication - System V

Three major new features were introduced in System V.

The first was *shared memory*. Shared memory is much faster than pipes, because there is no copying of data. Facilities exist to control access to shared memory, as well as synchronized updating by multiple processes.

However, there is one deficiency of shared memory in System V: processes using it must be *related* by a common ancestor.

This restriction does not exist for the second major new feature, *message queues*. Message queues provide a way for unrelated processes to share data. Messages may have types -- for example, a process may request all messages of a certain type from a queue.

The final addition in System V was *semaphores*. This well-known facility provided the much-needed, solid foundation for process synchronization.

At least two of these facilities, however (semaphores and shared memory) are heavily biased towards applications running on a single processor with all processes sharing the same memory space. In order to pursue its own research goals in the area of distributed computing systems, Berkeley introduced an entirely different set of facilities.

Interprocess Communication - 4.2 BSD

The facilities for interprocess communication have been enhanced at Berkeley to an extent unmatched in any other

Unix version. They represent an entirely different approach to interprocess communication.

The Computer Systems Research Group concluded that, in order to achieve its goal of truly distributed systems, the interprocess communication facilities should be *layered over networking facilities within the kernel itself*. In addition, the facility has been decoupled from the Unix file system and implemented as a completely independent subsystem[4].

In 4.2 BSD the *socket* is the building block for communication. Sockets are named endpoints of communication within a *domain*. Currently, the Unix and DARPA Internet communication domains are supported.

There are basically three types of sockets:

- *stream* sockets, which provide a reliable bidirectional flow of data to unrelated processes. In the Unix domain, pipes are available as a special case of sockets.

- *datagram* sockets, which provided facilities similar to those found in Ethernet.

- *raw* sockets, generally intended for use in the development of new communication protocols.

The Lowest Common Denominator

What do these four versions of interprocess communication have in common? Unfortunately, little more than pipes! And the simple byte stream as the lowest common denominator is very low indeed. Defining a set of system calls to handle only such restricted cases is simply not realistic.

We cannot expect developers programming advanced applications to give up the advanced facilities for interprocess communication discussed above and to accept of a standard based upon a more restricted set. The interprocess communication facilities of the various Unix versions must converge much more before a truly representative set of system calls can be defined.

Future Convergence?

The convergence of the AT&T Unix world towards System V will continue. Microsoft has reached an agreement with AT&T whereby the next release of XENIX will conform to the System V Interface Definition.

A Berkeley 4.3 distribution will reportedly be available soon -- officially "sometime before 1995" (this statement constitutes a reaction to the delays and disappointments surrounding the date of release of 4.2 BSD). The 4.3 release will, however, consist essentially of features and

enhancements that are necessary to stabilize problems that were identified in the current release.

In a major step towards reconciliation of the two major standards, Sun Microsystems and AT&T have agreed to work together to facilitate convergence of System V with Sun's 4.2BSD-based operating system.

They will attempt to merge the two standards into a single version. Technical representatives are meeting periodically to define a common applications interface. Sun intends to add complete compatibility with the System V Interface Definition, while maintaining the added functionality of 4.2BSD that was discussed above.

The Hopes for a Standard Unix Environment

Although the problems of standardization arising from the kernel incompatibilities discussed above are very real, there is a very large and important class of programs that are not affected by them. These are the software engineering tools used in the production of software, such as editors, report generators, filters, testing tools, document production facilities and compilers. Such programs are well served by the proposed standard in its current form. Thus we should expect that a large part of the Unix software engineering environment can indeed be standardized.

However, in advanced areas such as networking, distributed systems, and real-time systems, Berkeley Unix clearly offers superior facilities. Given the current nonconformity of Berkeley Unix to the standards pursued by most of the rest of the Unix world, a standard Unix software engineering environment will depend heavily on the success of efforts to merge these two worlds into a single one.

REFERENCES

[1] Proposed Standard, /usr/group Standards Committee, January 17, 1984. Obtainable from /usr/group, 4655 Old Ironsides Drive, Suite 200, Santa Clara, California 95050, U.S.A.

[2] AT&T System V Interface Definition, January 1985. Obtainable from AT&T Customer Information Center, Select Code 307-127, 2833 North Franklin Road, Indianapolis, Indiana 46129, U.S.A.

[3] PCTE: A Basis for a Portable Common Tool Environment, Functional Specifications, First Edition, August 1984, Bull, GEC, ICL, Nixdorf, Olivetti, Siemens.

[4] S.J. Leffler, R.S. Fabry and W.N. Joy, *A 4.2 BSD Interprocess Communication Primer*, Computer Systems Research Group, Dept. of EE&CS, Univ. of Calif. Berkeley, 1983.

FAULT TOLERANT SYSTEMS IN MILITARY APPLICATIONS

M. R. Moulding

Royal Military College of Science, Shrivenham, U.K.

Abstract. This paper introduces some basic principles and terminology associated with the design of highly reliable computer systems, and describes two experimental, fault tolerant computer systems which have been recently developed for military applications. The first system, called ADNET, demonstrates how a dynamically reconfigurable, local area network can be constructed so that various forms of hardware failure can be tolerated. The second system illustrates how software fault tolerance techniques can be used in a real-time application in order to cope with software design faults and, thereby, improve the software reliability of the system.

1. Introduction

Computer systems are often used to perform critical functions where their assured reliability is of paramount importance. Such applications have traditionally been associated with military and aerospace agencies which have funded much of the research into the development of highly reliable computer systems. The notion of incorporating protective redundancy into a system as a means of tolerating operational faults has emerged from this work to become a well-established technique for achieving high reliability. Such redundancy normally involves the inclusion of extra hardware units, additional software and spare processing time, and represents the price paid for increased reliability.

During the last decade, the reducing cost of hardware has led to the widespread use of computers in general industrial and commercial systems, and a growing number of these applications have stringent reliability requirements which demand the adoption of fault tolerance techniques. Consequently, there is an increasing need for industrial and commercial system designers to appreciate the architectural features of modern fault tolerant computers and it is the purpose of this paper to provide an introduction to this technology. In order to eliminate a possible source of misunderstanding, we shall first start with a brief discussion of some basic reliability concepts and terms.

2. Terminology

The field of computer systems reliability brings together a number of disparate professional groups (e.g. hardware designers, control engineers, software engineers) which each have their own set of terms to describe essentially the same reliability concepts. This can lead to a great deal of confusion and fruitless debate which prevents a meaningful discussion

of the underlying concepts. The reconciliation of this problem is beyond the scope of this paper but, in order to discuss some basic reliability issues, a consistent set of terms is required. We shall base our terminology on that provided by reference 1 in the knowledge that not all aspects of it are universally accepted.

Generally, the "reliability" of a system is characterised by a mathematical function R(t) which expresses the probability that a system will not fail during a specified time interval. For operational systems, this function cannot be known but by using reliability modelling techniques[2,3] its form may be predicted and the values of its parameters estimated. If such a modelling exercise can be carried out successfully, then it is possible to estimate the probability of a system failing during a defined operational period. This, however, is not a commonly used reliability metric since it can only be accessed via an appropriate model. Instead, mean time between failure (MTBF) is often used since it can be measured directly for an operational system by recording system failures, or obtained from a suitable model which itself may often require recorded failure data to refine its prediction.

Clearly, any attempt to assess objectively the reliabilty of a system requires a precise definition of what constitutes system "failure". For this we require a specification detailing the precise functionality of the system. A "failure" can then be said to occur when the behaviour of a system first deviaties from that defined by its specification. Such an innocuous definition belies the immense difficulty of producing a specification adequate for this purpose. The specification must be internally consistent; it must be complete in the sense that all possible operating conditions and responses are catered for; it must be authoratative so that judgements derived from it are unquestionable; and it must be expressed in a way which allows it to be used always as a test for system failure. A specification possessing these properties is termed an "exact specification" and since such specifications rarely exist we must accept that the definition of failure, and hence the assessment of reliability, will usually contain a degree of subjectivity.

Possibly two of the most frequently used reliability terms are "error" and "fault" and in order to allocate precise definitions for them, it is necessary first to construct a simple model of a computer system. Generally, we can consider a computer system to contain a hardware system and a software system which combine to perform the external functions of the total system. Each of these logically separate systems will contain a number of "components" which interact under the control of a "design". Components may themselves be considered as systems in their own right and contain sub-components interacting under a design to perform the overall functions of the component. This hierarchical decomposition will continue until "atomic" components, which are considered to have no internal design, are identified. In the case of a hardware system, decomposition will result in a series of designs and a set of physical (atomic) components; a software system will decompose purely into a set of designs, since software has no physical properties. During operation, a system will adopt a number of distinct internal states; for example, the values of program variables in a software system, or the bus voltage levels in a hardware system. When a computer system fails this will result from one or more defective values in the state of the hardware or software systems. These defective values are termed "errors" in the state of the system.

Such an error will itself be caused by either the failure of a physical component, a hardware design failure or a software design failure. Physical component failures are normally considered to arise from an ageing process which introduces defective values into the internal state of a component and eventually causes it to operate outside its specification. Hardware and software design failures result from defective values in the state of a design; for example, a missing connection on a circuit diagram or an incorrect statement in the source text of a program. A defective value, either in the internal state of a component or in the state of a design, which causes an error in the system state, will be viewed as a "fault" in the system.

From the discussion above we can derive a relatively simple model for computer system failure. During normal operation, the activity of the system may be such that a physical component fault, or a residual hardware or software design fault, is encountered and damages the system state by generating one or more errors. Again, depending on the precise activity of the system, an error may cause the external behaviour of the system to deviate from its specification and so cause a system failure. There are two complementary techniques which can be used to reduce the possibility of system failure and so provide high reliability:

(i) Fault Prevention. The aim of this technique is to try to prevent faults from existing in an operational system. There are two separate approaches:

(a) Fault Avoidance. The objective here is to try to avoid the introduction of faults into the system. For example, design faults can be reduced by the adoption of a good design method; physical component faults can be reduced by the use of top quality components.

(b) Fault Removal. This method assumes that faults will have been introduced during the development of a system and strives to remove as many as possible, by exhaustive validation and testing, before the system is launched into service.

(ii) Fault Tolerance. If prevention schemes cannot provide the required reliability for a computer system, then it will be necessary to construct the hardware and software systems in such a way that they can prevent a fault from causing system failure. Such an approach requires a combination of the following activities to be carried out by the system:

(a) Error Detection. The presence of the fault cannot be detected directly; it is the detection of an error in the system state which identifies the presence of a fault and can be used to instigate corrective action.

(b) Damage Assessment. Following the detection of an error, the extent of the damage to the system state may be estimated. Such estimates are normally based upon static damage confinement structures which exist within the system.

(c) Error Recovery. Before a system can be allowed to operate

normally following the detection of an error, the system must be returned to an error-free state.

(d) Fault Treatment. If a system is allowed to continue normal service following error recovery, it is possible that the fault will recur and lead to eventual system failure. To avoid this problem it is necessary to locate the fault and remove it from the system, by some form of reconfiguration, before allowing normal service to continue.

In the following sections we shall investigate how the fault tolerance principles outlined above can be applied to hardware and software systems.

3. Hardware Fault Tolerance

Hardware fault tolerance schemes are invariably based on the assumption that hardware design faults will not exist and that physical component faults will be the sole cause of potential failures. The rationale for this lies in the lower complexity of hardware designs when compared with software, and the widespread use of standard, operationally proven designs for integrated circuit devices and printed circuit boards.

The ageing process of physical components can be well characterised via accelerated life testing and, consequently, the effects of a component fault are "predictable". This is of significant advantage when attempting to devise a fault tolerance strategy. For example, if a certain component is known to fail in a particular manner, then the resulting damage to the system state can be predicted thus facilitating error detection and recovery. Furthermore, redundant components which must be added to the system to protect against physical faults can be of the same type and design.

Redundancy in hardware systems can be categorised as either "dynamic" or "static"[4]. In a dynamic scheme, a faulty component will usually provide some level of assistance with error detection but will rely on its surrounding environment to carry out the other phases necessary for fault tolerant operation. The standby sparing scheme illustrated in slide 4 is an example of such an approach. Here a main system component (M), periodically reports to a "watchdog timer" (W). Should the main component fail to report within a specified time interval then this will be recognised as an error by the timer which will be responsible for assessing the damage caused by the component fault, recovering the system to an error-free state and treating the fault by switching in a redundant "standby" (S) component. The damage assessment and error recovery phases can vary in sophistication. A simple approach is to assume that the entire state is damaged and to recover it by means of a hardware reset. More elaborate strategies can lead to resets for only parts of the system, based on some a priori characterisation of the fault and/or damage confinement structures which exist in the system.

In contrast to the dynamic redundancy approach where the surrounding environment of a component plays an important role in the overall fault tolerant behaviour, the objective of static redundancy is to mask the effects of a component fault from the surrounding environment. The canonical example of static redundancy is the triple modular redundancy

(TMR) unit illustrated in slide 4. Here, three identical components are subjected to the same inputs and the overall output is obtained by a two-out-of-three vote on the outputs of the individual components. Consequently, the TMR unit will mask the effects of any single component fault. Error detection is provided by the voting check which also locates the faulty component. Damage assessment is based on the assumption that the faulty component operates in complete isolation (termed an atomic action) and, consequently, cannot damage the system state. Error recovery simply involves ignoring the output values identified by the voting check as being erroneous; fault treatment may involve ignoring future outputs from that component, depending on whether the fault is considered transient or not. Where future output from a faulty component is ignored then the TMR unit will lose its fault masking properties unless a new component is switched in to replace the faulty one.

In principle, redundancy can be applied at any level within a system. However, the higher the level at which it is applied, the larger the range of faults it protects against. The reducing costs of hardware and the increasing functionality of integrated circuits mitigates against the traditional cost penalties of this approach and a number of fault tolerant multi-processor systems have been developed (e.g. references 5,6,7,8) where redundancy is applied at the intra-computer bus level (e.g. processor and memory modules). The emergence of local area network technology also invites the application of redundancy at the inter-computer bus level (i.e. a local area network with redundant computer systems). Where redundancy is applied at the processor or computer level the fault tolerance behaviour is usually controlled by software.

In the military domain, the notion of fault tolerant local area networks is of particular interest since the overall effect of such an approach is to distribute geographically the redundancy. This results in a system which is tolerant to both operational faults and action damage (faults generated by local environmental changes!). In the following section, we shall examine an experimental, fault tolerant local area network, called ADNET[9], which has been developed by the U.K. Ministry of Defence for application to shipborne command and control systems of the Royal Navy.

4. The ADNET Experiment

The fighting capability of present day warships is controlled by a substantial and closely integrated team of officers and their supporting staff. The team must derive a continuous and rapid assessment of the situation in the vessels area of concern from a confusion of data available to them from own ship's sensors, data links and many other sources. Based on this assessment, decisions must be made regarding the deployment of weapons. Once deployed, these resources must be controlled in order to achieve the desired effect. The size, complexity and time constraints of these tasks demand substantial computer assistance and this is provided by the command and control system.

The command and control systems of current RN warships are based on a centralised computer architecture, as illustrated in slide 5. The central computer supports not only the command and control functions required by

the officers and their staff but also the data processing and control requirements of the ship's sensors and weapons. Such an architecture lacks enhancement capability, and is vulnerable to action damage. Consequently, a futuristic command and control system architecture has been proposed in which each weapon and sensor has computer power for local data processing, and the command and control functions are horizontally distributed across a number of computers, each with their own operator display. The proposed configuration, illustrated in slide 6, uses a serial data highway to provide the necessary inter-computer communications.

In order to investigate the feasibility of a horizontally distributed command and control system, the ADNET experimental model was developed at the Admiralty Surface Weapons Establishment (ASWE). A simplified schematic representation of the system is illustrated in slide 7. At the heart of the system is the ASWE Serial Highway[10] which is a multi-drop bus to which access is by poll and response under the control of a highway controller. Some principal features are as follows:

3 Mbit/sec signalling rate, giving a maximum useful data rate of 1.8 Mbit/sec.

An upper limit of 63 nodes over a total highway length of 300 metres.

Intelligent communication processors which interface directly into the host computer's memory and handle all the highway level data transfer protocols. The host software simply views the highway as a high integrity memory-to-memory data transfer medium.

Broadcast and point-to-point messages of variable length up to 64 bytes.

Block data transfer up to 16k bytes.

Multiple levels of error detection with subsequent recovery of lost or corrupted messages (performed automatically by the communication processor without host involvement).

The use of a passive multi-drop connection to the highway means that communications in the network will not be affected by the powering down of any computer node. The use of a highway contoller does, however, represent a potential source of network failure and, to overcome this, the highway is equipped with two controllers. At any particular time, one controller will act as "master" and control the ordering of messages onto the highway in the normal manner; the other "slave" controller will monitor the activity of the highway and so be able to detect the failure of the master. If the master does fail, then the slave recovers the network to a consistent state and assumes the master function. This is a dynamic redundancy, standby sparing scheme where the standby unit performs a watchdog function. The physical cabling of the highway is also vulnerable to action damage and, in order to protect against this source of potential failure, redundant cabling is used. Each communications processor is connected to a number of cables (typically three). It transmits on all cables and selects one of that number as its input. If the reception on one particular cable has an unacceptably high error rate,

then the communications processor will automatically select another cable for input. Again, this is a dynamic redundancy scheme where the communication processor is responsible for implementing all phases of fault tolerance.

Connected to the highway in ADNET are a number of Ferranti Argus computers which collectively provide the command and control functions of the system via appropriate operator diplays (not illustrated). A sensor simulator computer is also connected to the network (not illustrated) in order to provide data for the command and control functions. All ADNET software is based on MASCOT[11] and programmed in CORAL.

Although the highway itself is capable of fault tolerant operation, fault tolerance at the system level will be identified by the preservation of the command and control functions in the presence of the failure of a particular computer node. This can be achieved by adding redundant computing power to the network and providing the software with the ability to reconfigure itself dynamically in the event of the failure of a particular node.

The ADNET approach to the dynamic reconfigurability problem is based on the concept that each computer should operate largely autonomously, rather than forming a partition of an integrated system which is managed by a global executive. The autonomous approach is reflected in the inter-process communications philosophy[12] which does not rely on fixed connectivity tables but instead allows a distributed command and control function to establish dynamically its own communications links across the network. One important aspect of remote process communication is based on a "user-service" model in which "user" processes require to access resources provided by remote "service" processes. Initially, when a user first requires to access a resource, it will broadcast an enquiry message onto the highway which will be received by all services of the specified type. These services will respond directly to the user by means of a point-to-point message, the destination of which was contained within the user's original request. If more than one reply is received, then the user will select the most appropriate service, possibly from status information contained within the reply. Thereafter, whenever the user wishes to access the service, it will do so via a point-to-point message, the destination of which was embedded within the service's reply to the original broadcast enquiry. The reply from the service will be a point-to-point message in the same way as before. If a computer containing a service fails, then all users of that service will time-out the point-to-point replies from it and, by the broadcast enquiry technique described above, establish a connection to a similar service held elsewhere.

In fact, the protocol described above is one of a related set supported by a communications package resident in each computer. The communications packages present an integrated view of the network such that processes in the same machine communicate in the same way as if they were remote. Consequently, processes can migrate around the network and automatically re-establish their required connectivity, regardless of the particular computer they, or their communicating partners, may be resident in. This characteristic means that, providing the services that a computer supports are replicated elsewhere in the network, a particular machine may be powered down, taken out of service, powered up, reloaded with new software

and then introduced back into the network, without disrupting the operation of the other machines. Such behaviour provides great flexibility in the run-time re-allocation of computer functions and significantly aids on-line maintenance.

The simple user-service model described above reveals the basic ADNET fault tolerance approach. Protective, dynamic redundancy is introduced by replicating services, which of course implies the provision of spare computing power in the network. Error detection is performed by user processes timing-out service replies. Damage assessment is based upon the physical structuring of the system since it is implicitly assumed that damage will be contained within the failed computer. Error recovery is limited to a user process disregarding any partial results obtained from the service before it failed. Fault treatment is performed by the user when it dynamically links itself to another version of the service held elsewhere.

If a service process is memoryless in the sense that its function does not depend on data retained between invocations (e.g. a mathematical function), then replicating it presents little difficulty since all service replicas will provide exactly the same function. However, if the service does retain data, then the replicas must be synchronised in some way in order that they offer the same service at all times. In ADNET such services are integrated into a specially developed, distributed database management system[13] which controls the replication, synchronisation and distribution of database partitions.

5. Software Fault Tolerance

In contrast to hardware fault tolerance where only physical component faults are usually considered, software fault tolerance schemes are concerned solely with design faults. This has two important ramifications:

(i) The faults and their effects will be "unpredictable". This increases the difficulty associated with error detection and recovery phases of fault tolerance. Backward recovery to a prior, error-free state is the most effective way of recovering from unpredictable faults.

(ii) Protective redundancy must be based on modules of independent design so as to minimise the possibility of common design faults.

The two main techniques for software fault tolerance are recovery blocks[15] and N-version programming[16]. The general syntax of a recovery block is illustrated in slide 8. A number of alternate modules of independent design are produced from the same specification. There will exist a primary alternate which represents the preferred design and a number of other alternates. These may be older versions of the primary (uncorrupted by enhancements), modules offering degraded functionality, or simply alternates providing the same functionality as the primary but based on different algorithms and/or produced by separate programming teams. On entry to a recovery block, a recovery point is established which allows the program to restore to this state, if required. The

primary alternate is executed and an acceptance test checks for successful operation. If the acceptance test fails, then the program is recovered to the recovery point taken on entry to the recovery block, the secondary alternate is executed and the acceptance test applied again. This sequence continues until either an acceptance test is passed or all alternates have failed the acceptance test. If the acceptance test is passed, then the recovery point taken on entry is discarded and the recovery block is exited. If all alternates fail the acceptance test, then a failure exception will be raised. Since recovery blocks can be nested, then the raising of such an exception from an inner recovery block would invoke recovery in the enclosing block. Generally, an exception raised from within an alternate can be used to indicate premature failure and thus invoke the same action as for an acceptance test failure.

The recovery block approach is essentially a software analogue of the hardware standby sparing scheme described in section 3. Redundancy is achieved by alternates of independent design; error detection is provided by the acceptance test or by an exception being raised from within an alternate. Ostensibly, damage assessment is not required because backward error recovery will eliminate all damage to the program. However, in a multi-processing environment, backward recovery will only be applied to a single process (or at most a defined set of interacting processes - see next section) and thus practical schemes will require protection mechanisms within the machine to confine the damage to that part of the system which will be backward recovered. This constitutes implicit damage assessment. Fault treatment within a recovery block is achieved by the execution of another alternate following recovery.

In contrast to recovery blocks, the N-version programming scheme, illustrated in slide 8, is a software analogue of hardware triple modular redundancy. Three or more (N) independently designed versions of a module are activated by a "driver" module (D) which supplies them with the appropriate input data. The driver then collects the individual outputs from the versions and performs a majority vote in order to determine the output from the N-version unit. Consequently, a design fault in any one module will be masked. Error detection is provided by the voting check which also locates the faulty version. Damage assessment is based on the premise that each version executes atomically (in isolation); this can be achieved physically by running each version on dedicated hardware or, logically, by running the versions on the same computer and using appropriate protection mechanisms. With atomic execution, error recovery is achieved by the driver ignoring the output values identified by the voting check as erroneous. Fault treatment can be considered as simply ignoring the results of the version identified as being faulty.

In a recovery block scheme, all alternates are always available on entry to the block, regardless of previous faults. The rationale for this is that a design fault will only be uncovered by a rare combination of processing conditions which are unlikely to recur when the recovery block is next executed. For an N-version scheme the situation is a little more complicated. Unlike the alternates of a recovery block, all versions of an N-version unit are usually executed each time the unit is invoked. Consequently, they can retain data locally between invocations. This has the advantages of increasing the design independence of the versions (alternates of a recovery block must all access the same global data structures which limits their algorithmic independence) and reduces the

data which must be passed to a version upon invocation. However, if the versions do retain data, then a driver will not be able to re-use a version which has produced an erroneous output since its internal state might have become inconsistent with the others of the unit. If the fault tolerance properties of the N-version unit are not to be degraded under these circumstances, it will be necessary to provide some form of recovery of the internal state of a faulty version.

Each of the two software fault tolerance schemes described above has its own virtues. Generally, the N-version programming scheme is most appropriate to those systems which have replicated hardware for concurrent execution of versions, and for which voting checks can be easily constructed (this can be a non-trivial exercise since the versions must be of independent design and their "correct" outputs can vary). Recovery blocks are most appropriate for systems where hardware resources are limited and voting checks are inappropriate. A full discussion of the relative merits of the two approaches can be found elsewhere[1]. The remainder of this paper will concentrate on the practical problems of using recovery blocks in real-time applications, and describe a demonstrator system, recently constructed at the University of Newcastle-upon-Tyne, to investigate the use of recovery blocks in a naval application.

6. Application of Recovery Blocks to Real-time Systems

Although recovery blocks have been available in principle since the mid-1970's, they have not been widely used in practical real-time applications. Some anticipated problems associated with their use are as follows:

(i) Run-time overhead. Acceptance tests, backward error recovery and additional alternate executions all provide a run-time overhead. Although acceptance test and alternate execution overheads are fundamental to the scheme, special hardware can be used to minimise backward error recovery times. The feasibility of this approach has been demonstrated at Newcastle University where a prototype "recovery cache" device[14] has been developed which backward recovers the memory of a DEC PDP 11/45. The overall configuration of the device is illustrated in slide 10. The recovery cache is based around a DEC LSI/11 microcomputer which communicates with the PDP 11/45 host processor via a cache-host interface unit (CHIU), and can access the memory of the PDP 11/45 via a cache-memory interface unit (CMIU). The host Unibus is physically intercepted by a bus monitor unit (BMU) which is controlled from the LSI/11, and which can write data directly to the recovery cache memory via a non processor request module (NPR). Under the conditions when the host processor does not require a recovery point, the BMU allows all host memory accesses to proceed unhindered. When the host instructs the cache to establish a recovery point, the LSI/11 configures the BMU to intercept all writes to memory locations which are being updated for the first time since the recovery point was established. Before these writes are allowed to proceed, the BMU reads the original value of the location and stores the location address and original value in the cache memory. It then applies the write to the host's

memory. If the host instructs the cache to recover, then the LSI/11 will read the address/value pairs from its memory and restore the appropriate locations of the host's memory to their original values. In this way, the memory is returned to its state when the recovery point was established.
The operation of the cache is determined by software which runs on the LSI/11 and, in its original form, this supports four levels of nested recovery points for a single process running on the host. Initial experiments with the cache indicated that, for a typical process, the run-time overhead of monitoring the Unibus was of the order of 10%.

(ii) Concurrent processing. When a regime of communicating processes establishes recovery points independently, then it is possible that the "domino effect"[15] will occur. This is illustrated in the first diagram of slide 10 where the horizontal lines describe the progress in time of two processes P1 and P2, the vertical lines indicate communication between processes and the open square brackets correspond to the establishment of recovery points. If, at the most advanced stage of its progress, P1 wishes to recover to its last recovery point, then this can be achieved without affecting P2. However, if process P2 wishes to recover to its last recovery point, then this will cause recovery beyond a communication with P1. In general, this communication must now be considered invalid (e.g. P2 may have passed P1 erroneous data) and so P1 must recover to its penultimate recovery point. In so doing, this invalidates further communication and causes P2 to recover to its penultimate recovery point. This sequence will continue until either a consistent pair of (possibly ancient) recovery points are found, in which case the system may proceed, or the processes will be left in an inconsistent state when all recovery points of one or both processes have been used up.
The general solution to the domino effect is to establish "recovery lines" in the system, as illustrated by the broken lines on the second diagram of slide 10. A recovery line connects a mutually consistent set of recovery points and can be achieved by groups of processes cooperating to form "conversations"[15]. On entry to a conversation, a process establishes a recovery point and, thereafter, may only communicate with others that have also entered the conversation. If a process wishes to recover whilst in a conversation, then all other processes of that conversation are forced to recover also. When a process wishes to leave the conversation, it must wait until all other processes are ready to leave. This, of course, introduces a synchronisation overhead but is the price paid for controlled recovery. Conversations, like recovery blocks, can be nested, as illustrated in slide 10. Here processes P1-P4 initially enter an outer conversation. Some time later, P1 and P2 form an inner conversation which, after two communications, completes and returns P1 and P2 to the outer conversation. At some future point in their processing, P1-P4 will synchronise to complete the outer conversation.
Although conversations provide a solution to the domino effect, the ease with which they can be implemented and used in practical systems is largely unknown, and the synchronisation overheads associated with their use is likely to be application dependent.

(iii) Acceptance tests. The acceptance test provides the basic method of error detection and, as such, plays a vital role in determining the overall effectiveness of the scheme. If the acceptance test is too complex, then it will generate a large run-time overhead and is liable to contain residual design faults. In contrast, a simple acceptance test may not provide an adequate method of checking the acceptability of an alternate's operation. Importantly, there is no wealth of documented practical experience upon which a designer of acceptance tests can draw.

(iv) Location of recovery blocks. For the effective utilisation of redundancy, recovery blocks should be used in those sections of the software most likely to contain faults which would cause system failure. The unpredictable nature of software faults makes this task extremely difficult.

(v) Development cost overhead. Software development overheads resulting from the use of recovery blocks can be divided into a fixed part and a proportional part. The fixed part will arise from the need to provide additional run-time environment software to support the operation of recovery blocks and conversations. The absence of a standard environment to provide this facility adds significant cost risk for any project contemplating the use of recovery blocks. The proportional part of the cost will be derived from the design and implementation of acceptance tests and redundant alternates, but will also include effort associated with selecting the locations of the recovery blocks and in defining suitable conversation structures. Again, there is little empirical evidence upon which to base estimates for these.

(vi) Memory overhead. Extra memory will be required for additional run-time support software, acceptance tests and redundant alternates. Such overheads are difficult to predict in the absence of practical experience.

(vii) Reliability improvement. If a system designer is prepared to argue for the inclusion of recovery blocks, what sort of reliability improvement, if any, can he expect to get? The risks and costs are evident; the benefits are unproven.

The circularity of the case against recovery blocks is manifest: recovery blocks have not been chosen for use in practical systems because there is insufficient evidence of their utility; there is insufficient evidence of the utility of recovery blocks because of their lack of use in practical systems. In an attempt to break free from this loop, a project, sited at Newcastle University, has recently been completed which has investigated the costs and benefits of using recovery blocks in a realistic, real-time system[17]. The work was funded jointly by the Ministry of Defence and the Science and Engineering Research Council of the U.K., and was directed at the construction of a demonstration system which modelled a subset of the functions of a centralised naval command and control system, as illustrated in slide 5.

The demonstration system consisted of three interconnected DEC computers, as shown in slide 12. The command and control software, in which recovery

blocks were included, was written in CORAL and based on MASCOT[11]. This ran on a DEC PDP 11/45, to which was connected the recovery cache described above, and a command console via which an operator could invoke command and control functions. The command and control machine was connected, by a parallel link, to a Unix-based PDP 11/45. This acted as a file-server on which monitoring output from the command and control machine was logged. The actions of own ship's sensors and weapons were simulated by MASCOT/CORAL software running on an LSI/11. Simulation scenarios were stored on the file server and read via a serial link. A graphics console was provided to allow an operator to control the operation of the simulator and to display the current state of the simulation. Communication between the command and control software and the simulated weapons and sensors was achieved via messages passed across a serial link.

The functionality of the command and control software was based upon anti-submarine warfare scenarios in which an operator would guide a torpedo-carrying helicopter to engage a hostile submarine. The command and control software was constructed in such a way that the software fault tolerance embedded within it could be either enabled or disabled. By running the command and control software in these two modes for various scenarios, comparative overall MTBFs could be obtained. Moreover, by using the monitor output from the command and control software, the fault coverage provided by the software fault tolerance could be estimated by determining the number of potential failures which were averted.

An important aspect of the work was the development of a scheme to apply the conversation principle to MASCOT software: a set of concurrent processes, termed activities, which interact through Inter-activity communication Data Areas (IDAs). The approach adopted was to define, at system construction time, static conversation structures called "dialogues"[18]. Each dialogue was created with a unique name, nest level (since dialogues, like conversations, may be nested), activity list (to define those activities which are permitted to use the dialogue) and IDA list (those IDAs via which dialogue activity members are allowed to communicate). Each activity is created with a set of dialogues which it may use; dialogues may be entered or exited and this is essentially the way an activity establishes and discards a recovery point explicitly. A recovery block called by an activity will be passed the dialogue name to be used when establishing the recovery point of the block.

The principle of static, named, dialogue structures is important since it provides good design visibility of the intended recovery structure. One major problem with the use of backward error recovery is associated with non-recoverable interfaces. Consider the situation where an activity fires a missile from within a recovery block. If the activity recovers for some reason (e.g. the acceptance test fails), then the internal state of the activity will be inconsistent with the state of its environment, since we cannot reverse time in the real world. One approach would be to insist that an activity never accesses a non-recoverable interface when it has an active recovery point. In conversation-type schemes, this can lead to excessive synchronisation overheads associated with the completion of conversations. In the dialogue scheme, this problem is avoided by distinguishing between forward and backward recoverable IDAs. Forward recoverable IDAs represent non-recoverable interfaces; backward recoverable IDAs are interfaces between activities within the recoverable

system. When a dialogue recovers, all associated backward recoverable IDAs are recovered in the normal manner; forward recoverable IDAs are not. Instead, a forward recovery procedure, which is specifically defined for that IDA, is executed. This will attempt to place the non-recoverable environment in a state consistent with that of the recovered activities. For example, in the case of a missile firing, the forward recovery procedure might self-destruct the missile and then decrement the (backward recovered) missile count by one.

The implementation of the dialogue scheme involved adding recovery software to the MASCOT run-time kernel to support recovery blocks and dialogues, and enhancing the recovery cache software to accommodate concurrent MASCOT activities. Some 3000 man-hours of effort was expended in this work and the MASCOT run-time kernel size was increased by approximately 25%.

The results of reliability measurements on the demonstrator system[17] indicated that approximately 70% of software failures were averted by the use of software fault tolerance and the MTBF increased by about 135%. In fact, around 90% of all command and control software faults were successfully detected but hardware faults in the prototype recovery cache, and residual bugs in the recovery software of the MASCOT kernel, prevented successful recovery. In the absence of such deficiences (which one would expect for standard, re-usable hardware and software), an increase in MTBF of 900% was predicted.

The price paid for this increase in reliability was as follows[17]:

(i) 60% increase in the cost of developing the command and control applications software;

(ii) 33% extra applications code was produced;

(iii) 35% extra applications data memory was required;

(iv) 40% additional run-time was required (30% dialogue synchronisation, 8% cache bus monitoring).

6. Conclusions

Since the 1950's, fault tolerance has been used to improve the reliability of hardware systems. The reducing cost of hardware and the increasing functionality of integrated circuit devices has led to the development of fault tolerant multi-processor and local area network systems where protective redundancy is applied at the processor and computer level, respectively. The inclusion of redundant computers in a local area network is particulary attractive in military applications since the geographical separation of the redundancy can lead to a system which is tolerant to both operational faults and action damage. This paper has described the essential features of a dynamically reconfigurable, local area network, called ADNET, which has been specifically designed to exploit these potential benefits for a distributed naval command and control application.

Traditionally, fault tolerance schemes have only considered the physical failure of hardware components, although it is often the case that computer system failures are the result of residual software design faults. Various software fault tolerance techniques have been proposed during the last decade but there has been little evidence of their widespread use in practical systems. However, an experimental system, recently constructed at Newcastle University, has demonstrated that software fault tolerance can significantly increase the reliability of real-time software, and an account of this work has been included in this paper.

The increasing complexity of hardware systems, and in particular the advent of VLSI devices of customised design, suggests that conventional assumptions regarding the absence of hardware design faults in systems can no longer be considered as generally valid. Consequently, it is likely that fault tolerant systems of the future will require redundant components of independent design to be added to hardware systems, in a similar manner to that currently proposed for software systems. Inevitably, the increasing ease with which we can implement computer systems exposes our inability to specify and design them correctly and, in the presence of such imperfection, we must become more tolerant!

References

1. T. Anderson and P. A. Lee, "Fault Tolerance: Principles and Practice," Prentice Hall, 1981.

2. W. G. Bouricius et al., "Reliability Modelling Techniques for Fault Tolerant Computers," IEEE Transactions on Computers, C-20(11), pp. 1306-1311, 1971.

3. P. A. Keiller, B. Littlewood, D. R. Miller and A. Sofer, "On the Quality of Software Reliability Prediction," Proc. NATO Advanced Study Institute on Electronic Systems Effectiveness and Life-Cycle Costing, Norwich, UK., 1982.

4. R. A. Short, "The Attainment of Reliable Digital Systems Through the Use of Redundancy - A Survey," IEEE Computer Group News 2(2), pp. 2-17, 1968.

5. J. H. Wensley et al., "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," Proc. IEEE 66(10), pp. 1240-1255, 1978.

6. A. L. Hopkins, T. B. Smith and J. H. Lala, "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," Proc. IEEE 66(10), pp. 1221-1240, 1978.

7. C. S. Repton, "Reliability Assurance for System 250, A Reliable, Real-Time Control System," First International Conference on Computer Communications, Washington (DC), pp. 297-305, 1972.

8. D. Katsuki et al., "Pluribus - An Operational Fault-Tolerant Multiprocessor," Proc. IEEE 66(10), pp. 1146-1159, 1978.

9. J. A. Gasden, "ADNET: An Experiment in Computer Networks for the Royal Navy," Proc. 3rd. International Conference on Distributed Computing Systems, 1982.

10. J. S. Hill and M. G. Stainsby, "A Highway for Intercomputer Communication," Journal of Naval Science, 6, 216, 1980.

11. MASCOT Suppliers Association, "The Official Handbook of MASCOT," RSRE, Malvern, U.K., 1980.

12. W. L. Lakin and M. R. Moulding, "The ADNET Communications System: Inter-Process Communication in a Fault Tolerant Local Network," Proc. Third IFAC/IFIP Workshop on Achieving Safe Real-Time Computer Systems, pp. 233-238, Cambridge, U.K., 1983.

13. P. R. Tillman, "ADDAM: ASWE Distributed Database Management System," Proc. 2nd. International Symposium on Distributed Database Management Systems, North Holland Publishing Company, 1982.

14. P. A. Lee, N. Ghani and K. Heron, "A Recovery Cache for the PDP-11," IEEE Transactions on Computers, C-29(6), pp. 546-549, 1980.

15. B. Randell, "System Structuring for Software fault Tolerance," IEEE Transactions on Software Engineering, SE-1(2), pp. 220-232, 1975.

16. L. Chen and A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," Digest of FTCS-8, Toulouse, pp. 3-9, 1978.

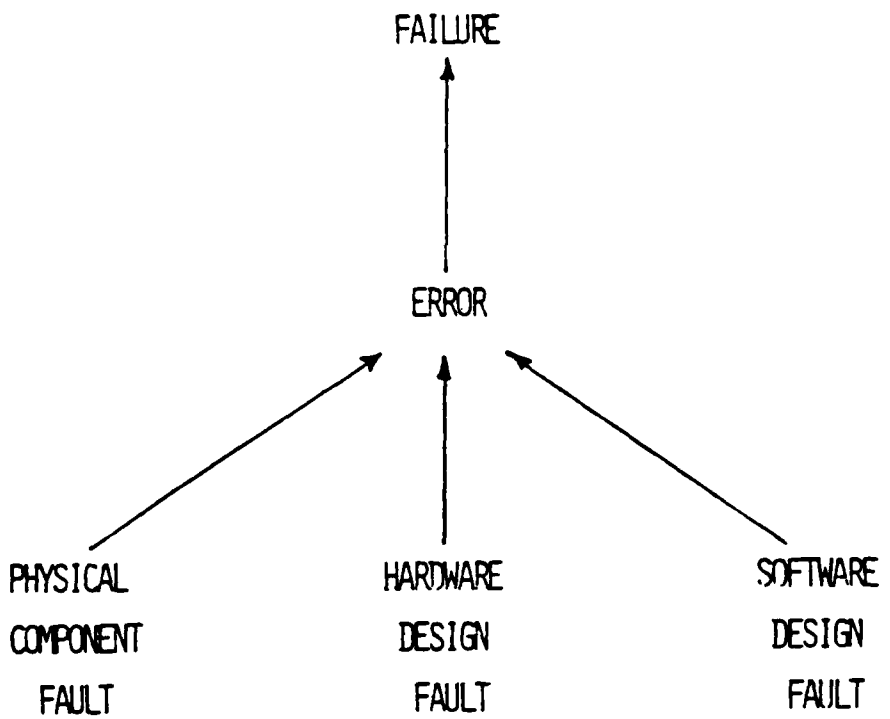
17. T. Anderson, P. A. Barrett, D. N. Halliwell and M. R. Moulding, "An Evaluation of Software Fault Tolerance in a Practical System," to appear in Digest of FTCS-15, Ann Arbor, 1985.

18. T. Anderson and M. R. Moulding, "Dialogues for Recovery Coordination in Concurrent Systems," In Preparation.

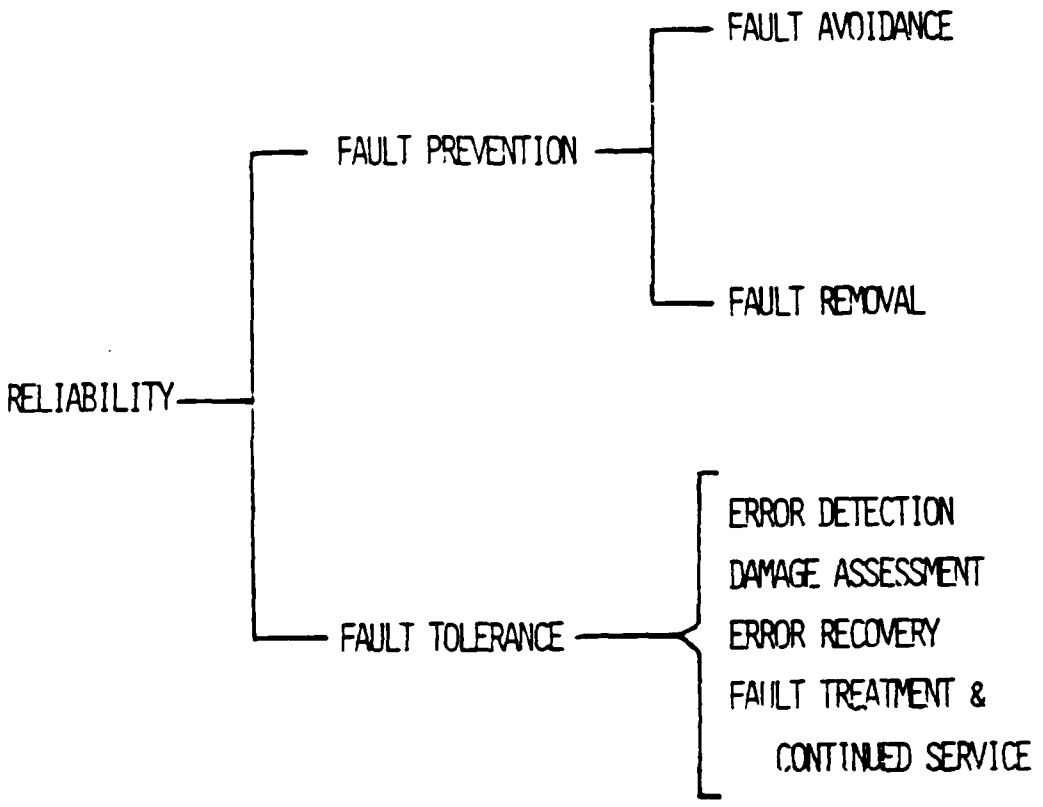
TERMINOLOGY (1)

RELIABILITY	IS CHARACTERISED BY A FUNCTION R(t) WHICH EXPRESSES THE PROBABILITY THAT A SYSTEM WILL NOT FAIL THROUGHOUT A PERIOD OF DURATION t
MTBF	MEAN TIME BETWEEN FAILURES
FAILURE	OF A SYSTEM OCCURS WHEN THE BEHAVIOUR OF THE SYSTEM FIRST DEVIATES FROM THAT REQUIRED BY ITS SPECIFICATION
EXACT SPECIFICATION	IS REQUIRED WHICH MUST BE: CONSISTENT COMPLETE AUTHORITATIVE USABLE AS A TEST FOR FAILURE
ERROR	A DEFECTIVE VALUE IN THE STATE OF A SYSTEM
FAULT	A DEFECTIVE VALUE IN THE INTERNAL STATE OF A COMPONENT, OR IN THE STATE OF A DESIGN.

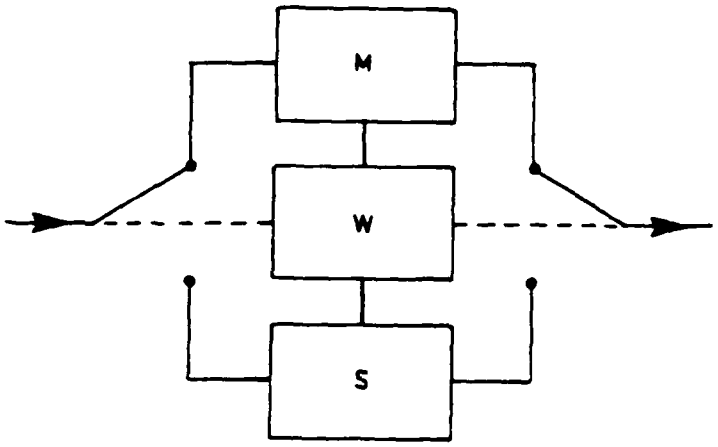
TERMINOLOGY (2)



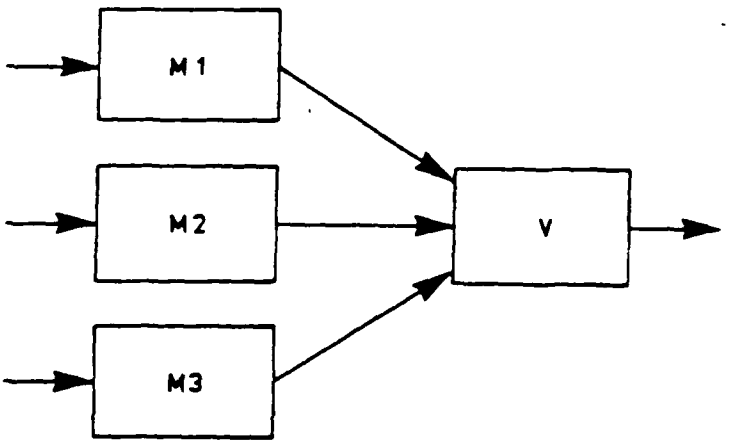
TERMINOLOGY (3)



HARDWARE FAULT TOLERANCE



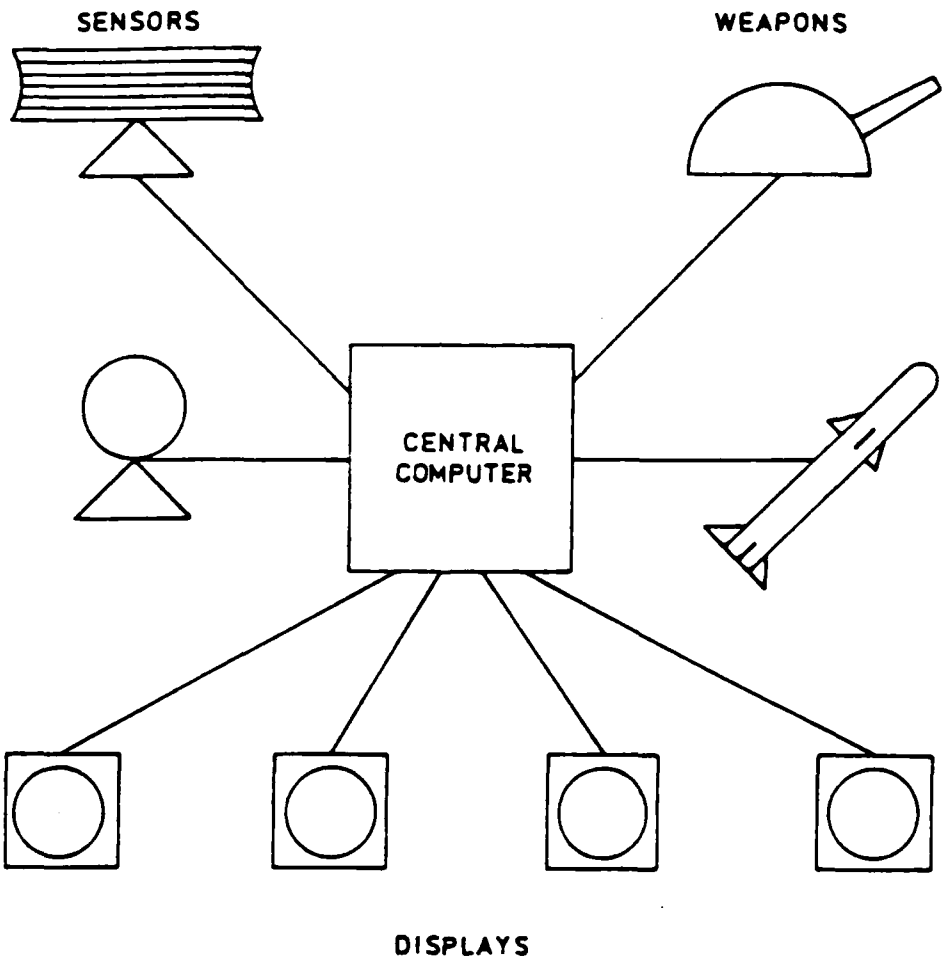
STANDBY SPARING
(DYNAMIC REDUNDANCY)



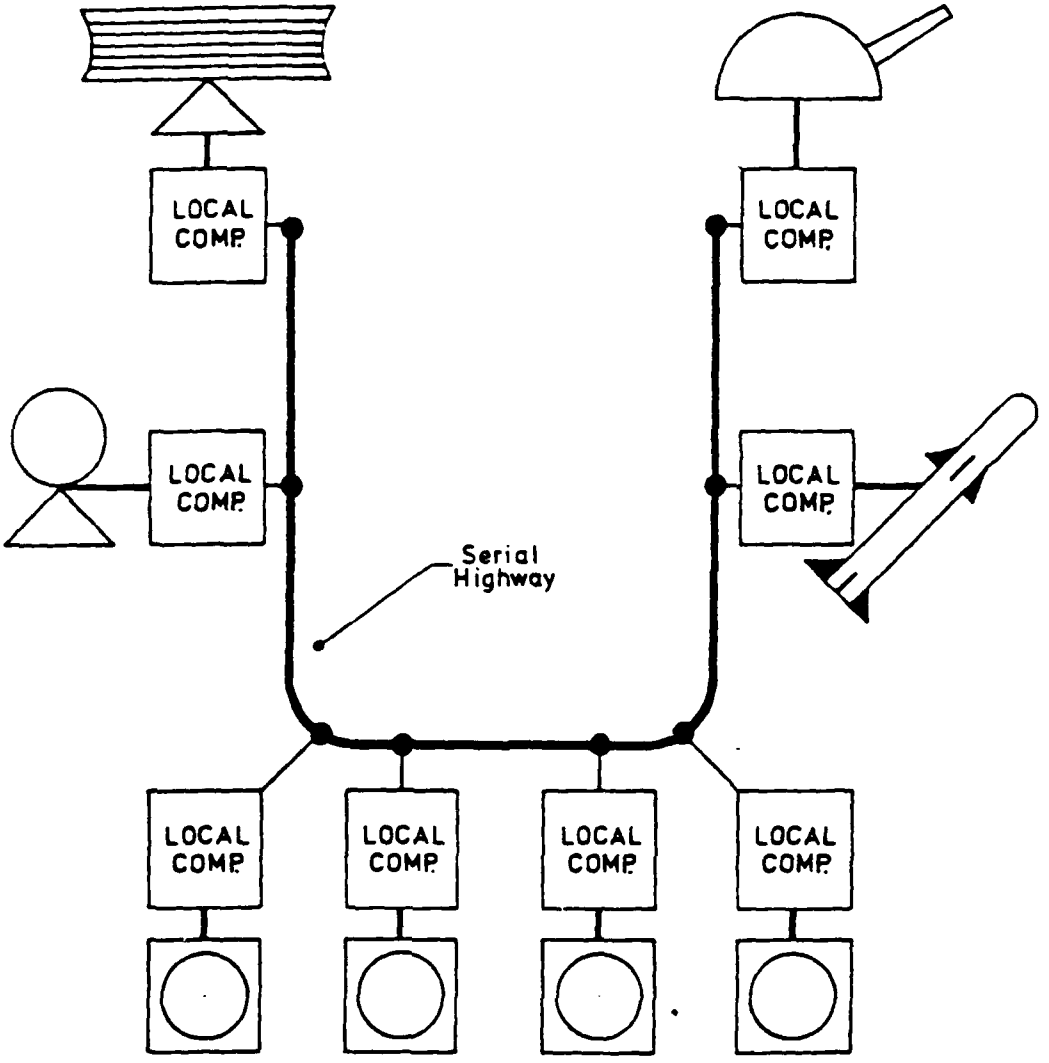
TRIPLE MODULAR REDUNDANCY
(MASKING OR STATIC REDUNDANCY)

RMCS — CRANFIELD

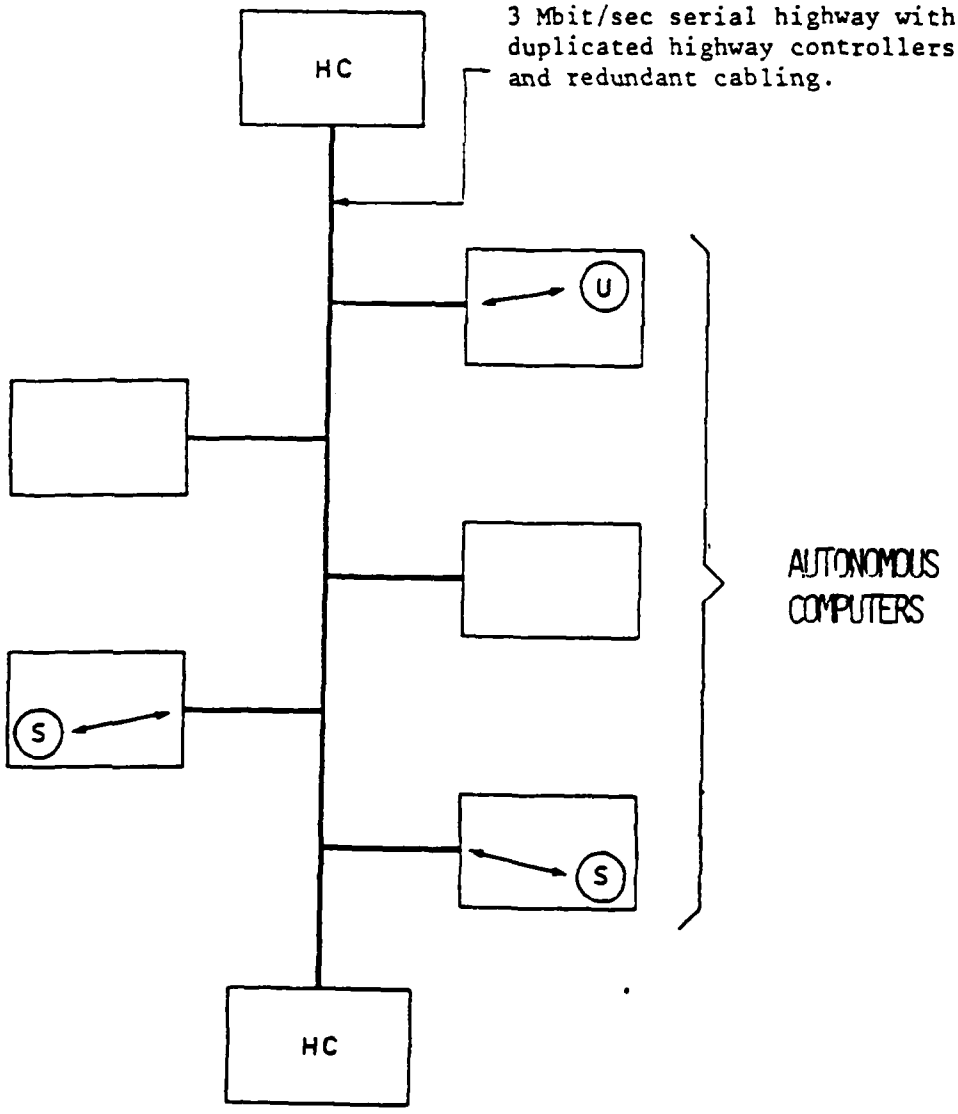
CENTRALISED COMMAND AND CONTROL SYSTEM



HORIZONTALLY DISTRIBUTED COMMAND AND CONTROL SYSTEM



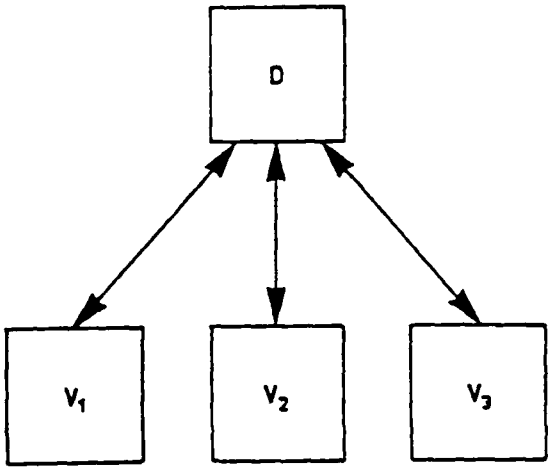
ACTION DATA NETWORK (ADNET)



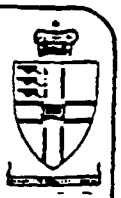
SOFTWARE FAULT TOLERANCE

ENSURE < Acceptance Test >
BY < Primary Alternate >
ELSE BY < Secondary Alternate >
.
.
.
ELSE BY < n^{th} Alternate >
ELSE ERROR

RECOVERY BLOCK



N-VERSION PROGRAMMING



USE OF RECOVERY BLOCKS

PROBLEMS ANTICIPATED:

1 RUN-TIME OVERHEAD

2 CONCURRENT PROCESSING

3 ACCEPTANCE TESTS

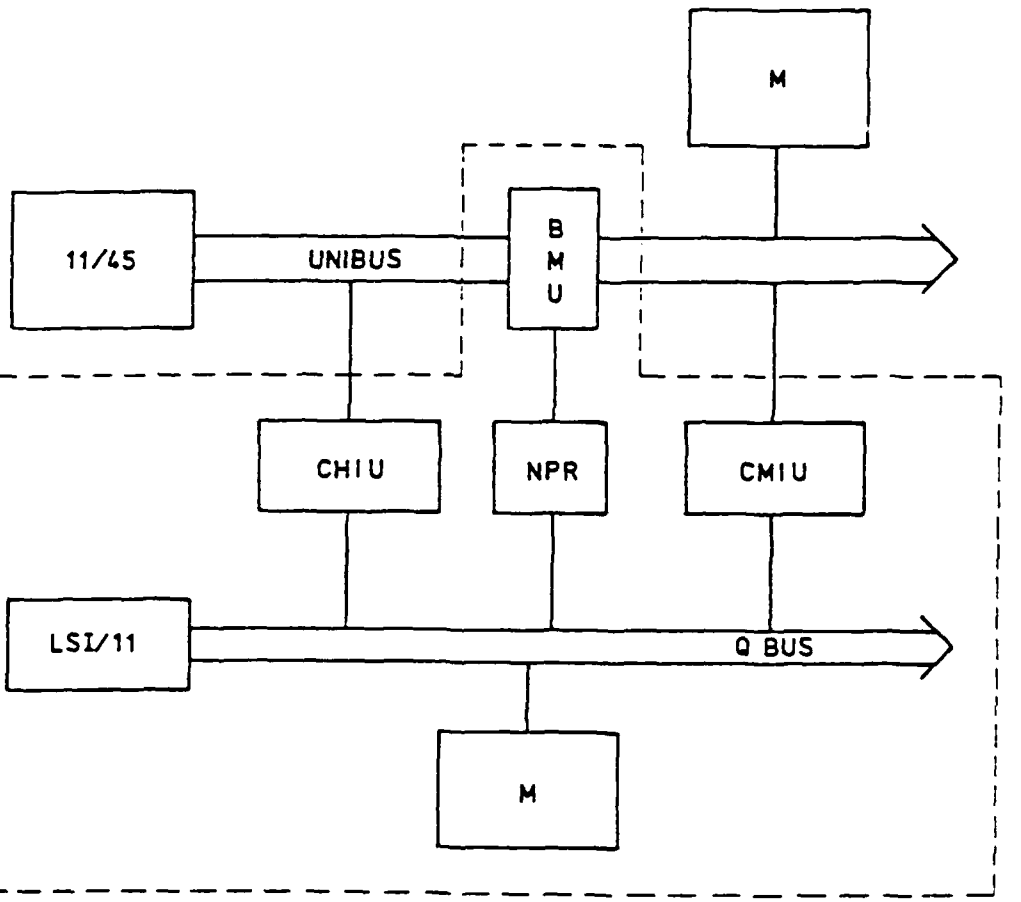
4 LOCATION OF RECOVERY BLOCKS

5 DEVELOPMENT COST OVERHEAD

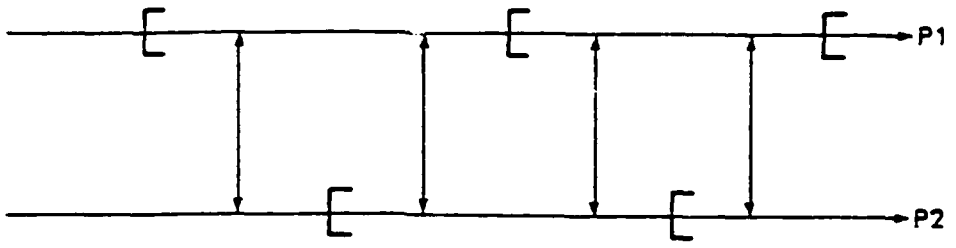
6 MEMORY OVERHEAD

7 IMPROVEMENT IN RELIABILITY?

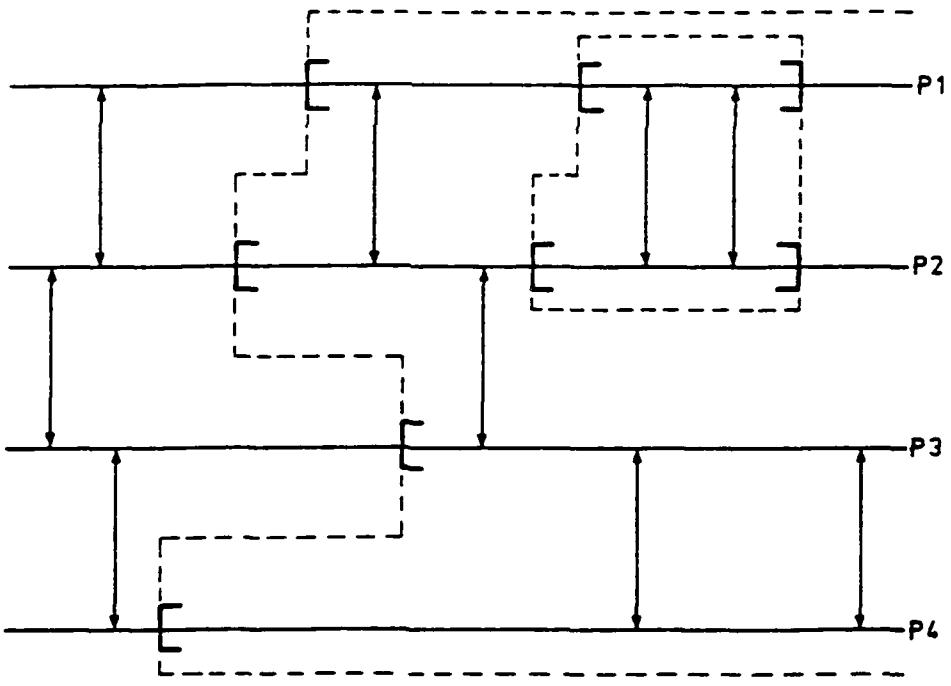
RECOVERY CACHE



RECOVERY BLOCKS AND CONCURRENCY

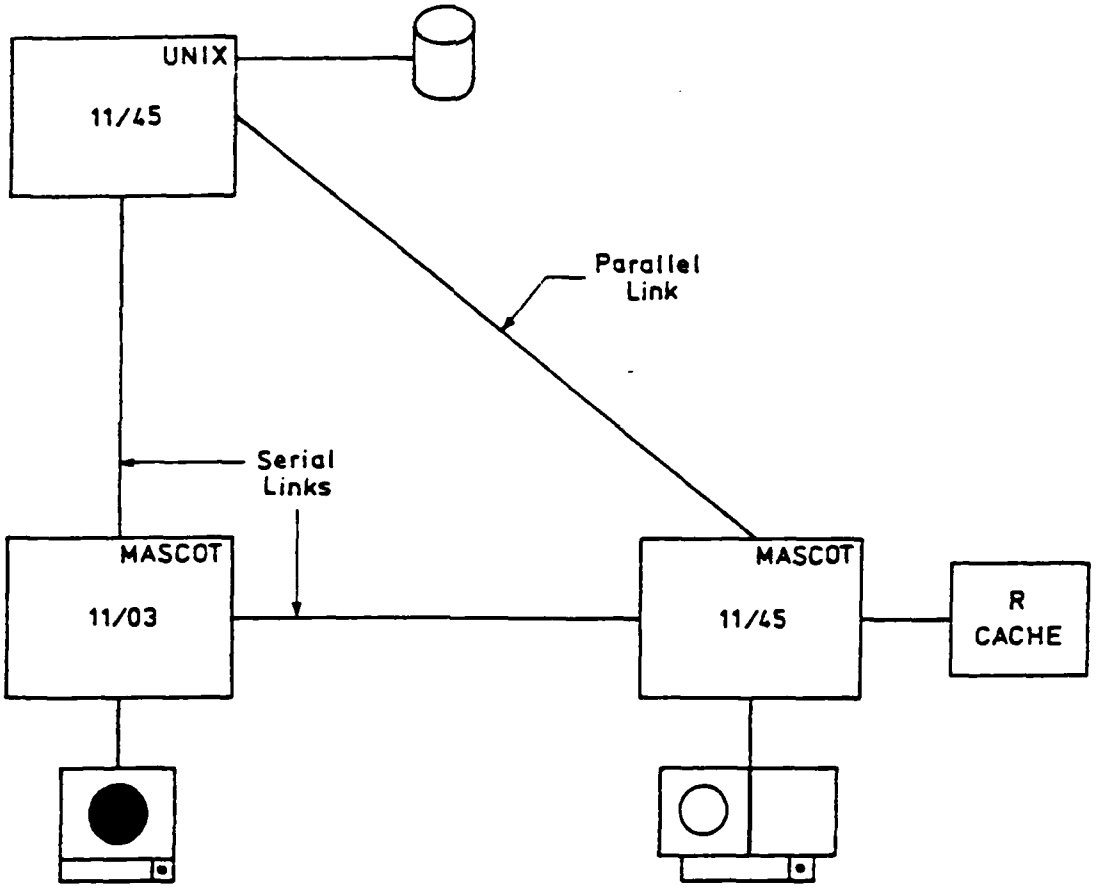


DOMINO EFFECT



RECOVERY LINES AND CONVERSATIONS

SOFTWARE FAULT TOLERANCE DEMONSTRATOR



EXPERIMENTAL OBSERVATIONS AND RESULTS

1 STATIC FORM OF CONVERSATION, TERMED A DIALOGUE, DEVISED FOR MASCOT

2 UTILITY OF DIALOGUES AND RECOVERY BLOCKS DEMONSTRATED

3 RELIABILITY IMPROVEMENTS

- 70% OF POTENTIAL FAILURES AVERTED
- 135% INCREASE IN MTBF

4 COSTS AND OVERHEADS

- 60% ADDITIONAL DESIGN & CODING EFFORT
- 33% EXTRA CODE
- 35% EXTRA DATA MEMORY
- 40% ADDITIONAL RUN-TIME
  - (30% SYNCHRONISATION OVERHEAD)

THE ASPECT PROJECT

J.A. Hall
Systems Designers plc

INTRODUCTION

ASPECT was the first Alvey-supported software engineering project and is a collaborative venture aimed at prototyping a multi-language, distributed-host, distributed-target Integrated Project Support Environment. The ASPECT team is led by Systems Designers plc (SD) and the other partners are the Universities of Newcastle upon Tyne and of York, GEC Computers Limited, ICL and MARI. Following the Alvey strategy, ASPECT has started by integrating existing tools, notably Perspective from SD, on UNIX ; this environment is being developed by distributing it using the Newcastle Connection and then building in the more advanced results of collaborative research and development.

REQUIREMENTS

To understand the objectives and strategy of ASPECT, it is necessary to consider the requirements for an IPSE : what we expect it to do, beyond what our current tools provide, to improve the software development process. We can identify four areas where an IPSE can advance the state of the art:

a) It must support the whole software lifecycle.

Whatever one's view of the software lifecycle, it certainly encompasses a number of phases through which the software progresses and, at every phase, a number of different types of activity : planning, managing, carrying out and recording the phase, for example. An IPSE, therefore, must support all these activities for every phase : more importantly, it must integrate the various supporting tools so they form a coherent whole.

b) It must support development methods.

Software development methods can be characterised by: their data model of software development ; the (frequently graphical) notation for expressing this model ; the rules which govern the application of the model and the procedures for manipulating it. An IPSE must be capable of supporting all these aspects of a method. Because there is no universal method and new methods are continually being introduced, an IPSE must be configurable, to support many methods, and capable of integrating different methods.

c) It must deliver power to the user.

We need to harness the raw hardware power now available so that both processing power and io bandwidth are more than enough for the user not to be constrained by the system. This implies that an IPSE must be workstation based and have an effective, responsive man-machine interface.

d) It must support development in the large.

An IPSE must support teams of people working on common projects. At the physical level this implies networking of machines ; at the logical level, version and configuration control, concurrency control, access control and task management must be built in to the IPSE.

KEY OBJECTIVES

ASPECT, in addressing these requirements, is concentrating on four key objectives.

a) Integration and openness.

ASPECT is emphasising the development of an infrastructure for tools, because it is by provision of a powerful set of common services to all tools that integration of tools can be achieved. Tools written for single users can immediately be used on large projects when incorporated into ASPECT, for example, because the infrastructure manages all the problems of controlled sharing between users. It is crucial that ASPECT provide these facilities in an open way so that new tools and methods can be incorporated by the user.

b) Host distribution.

ASPECT is addressed at developers who may be goegraphically distributed and who work on large projects using a range of machines including personal workstations.

c) Good man-machine interface.

A major part of the ASPECT research is aimed at providing an architecture in which software engineers can use the power of, for example, bit - mapped graphics and pointing devices effectively.

d) Target distribution.

ASPECT is oriented towards the development of embedded systems. In particular we are addressing the specification, development and testing of software for distributed target machines.

ASPECT ARCHITECTURE

The ASPECT architecture addresses, in a simple and general way, the key objectives. It is based on a clear separation between tools and kernel, and the provision by the kernel of a powerful set of common services for structuring and storing information, for communicating with users and other tools, and for manipulating remote targets. These functions are made available to tools through the public tool interface (PTI).

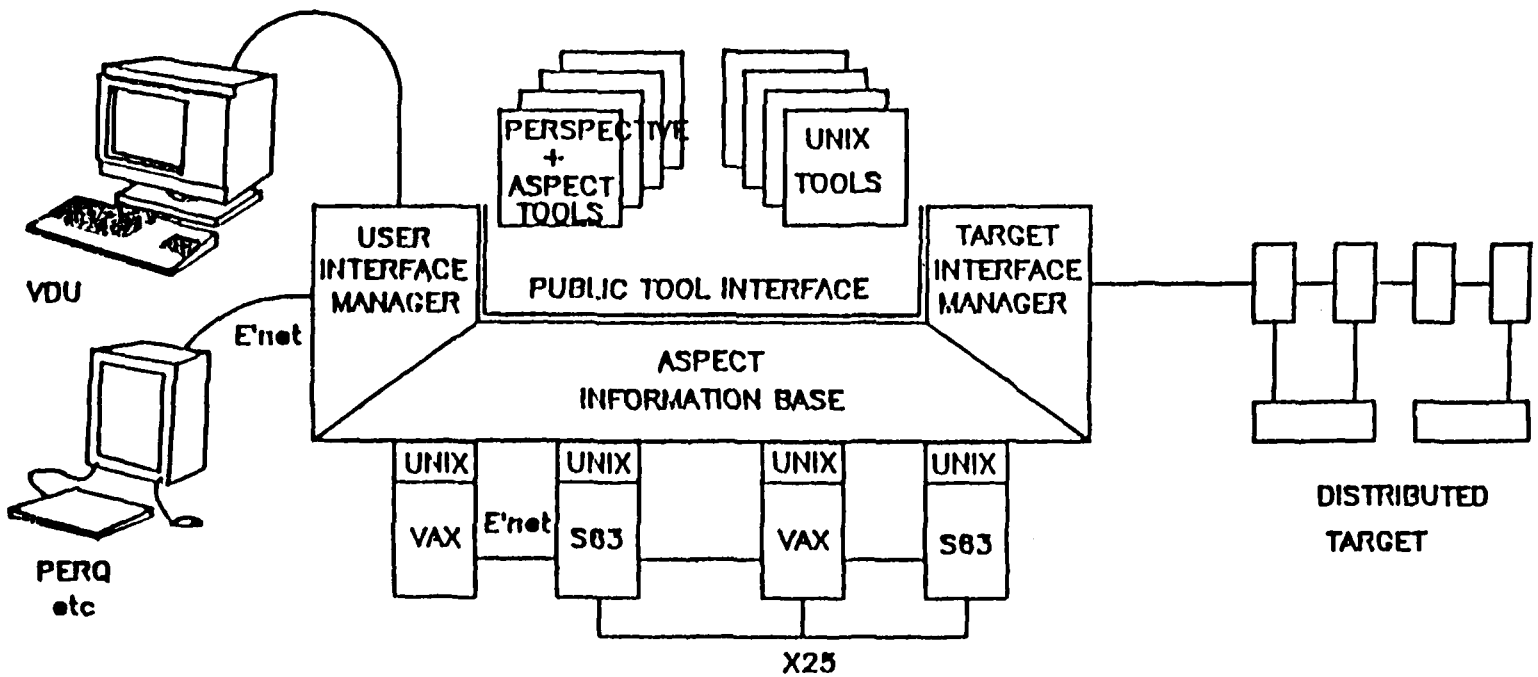
To achieve the required openness the PTI is extensible, to support new methods and tools, and configurable, so a project can impose particular methods of working, if required. Furthermore, since the PTI is the only means by which tools use ASPECT services, it necessarily includes within itself the facilities for its own extension and configuration.

One of the most important requirements on ASPECT is that existing tools, written for the host operating system, should be usable within ASPECT. This is made possible by the open tool interface (OTI). The OTI can be thought of as a subset of the PTI which appears to the tool just like the host operating system. ASPECT is hosted on UNIX*, so the OTI makes available to ASPECT a large collection of existing software development tools.

The PTI services fall into four groups:

- information storage
- man-machine interface
- process invocation and communication
- target services.

ASPECT ARCHITECTURE



Each of these services is provided by a layer of software - in the effect a subroutine library - between the tool and the UNIX Kernel. The PTI offers services at a much high level than those of UNIX. To provide the open tool interface, the PTI includes calls appearing to be UNIX system calls, but even these are processed by ASPECT rather than by UNIX so that all tools, including UNIX tools, are fully under the control of ASPECT. Indeed, different open tool interfaces will be provided for different UNIX tools to capture the semantics of the tools' data correctly in the ASPECT information base.

MEETING THE OBJECTIVES

The central component in ASPECT is the information base, and it is the information base which achieves the integration between tools by providing a central, structured repository for all the information they manipulate. The information base is a database but contains in addition:

a) Its own definition. The structural information can be accessed via the PTI in the same way as any other information.

b) Rules. These support not only the integrity constraints of the basic data model, but also user-defined rules which may, for example, be the rules governing the use of a particular development method.

c) Built in structures to support software engineering. In keeping with the aim of integration, many of the structures (for example version identification) supporting development in the large are provided at the information base level.

The database itself uses a standard architecture, the ANSI/SPARC three level model. This has a conceptual level, with below it an internal level and above it a set of external views, each view presenting the database to a tool in the way required by that tool. The conceptual level uses a standard data model, Codd's extended relational model called RM/T. This is a very powerful combination for providing the required extensibility and openness. The view mechanism not only shields tools from extension to the conceptual model, but it is powerful enough to transform the data to suit almost any tool. In particular the Open Tool Interface is achieved by defining UNIX-like views of the data.

The architecture of ASPECT is implemented on UNIX. In order to run ASPECT on a distributed host we are taking advantage of the Newcastle Connection, a powerful method of linking UNIX systems so that they behave as a single UNIX. ASPECT is building local and wide area networks of distributed UNIX systems. At a level above this, we are developing methods of distributing the information base and, in particular, supporting the logical distribution of the database between separate but interdependent users.

The mmi of ASPECT is, like the information base, aimed at providing a high level of functionality to tools and removing from individual tools concern with the details of user interaction. The mmi architecture has to achieve this across a wide range of users, of devices, of tools, and of interaction styles. At the same time the quality of interaction on powerful workstations is paramount. ASPECT handles this by defining levels of abstraction within the mmi and providing components, with well defined interfaces, to handle these abstractions. This is a major research topic in the project.

In approaching the programming of distributed targets, ASPECT is again looking for general, powerful solutions. We are studying methods for describing target architectures, extending languages to support interprocess communication on such targets, describing the placement of processes on processors and monitoring the operation of the target.

ASPECT STRATEGY AND THE ALVEY PROGRAMME

ASPECT is an Alvey second generation IPSE, in that it is clearly based on a database and is designed for a distributed host. It is not, however, being designed from scratch but is evolving from existing products and ideas. The main starting points were Perspective (a SD environment product), UNIX, and the Newcastle Connection. The initial release of ASPECT is indeed an integration of these components plus an Ada compiler. Meanwhile research has been going on to explore how to move forward from that base.

This research is now being brought together, with our experience from the first release, to define ASPECT - in particular its Public Tool Interface - in some detail. On the basis of this definition, prototypes of ASPECT will be produced and used as vehicles for research and further development. These prototypes will of course reuse as much as possible both of the partners' existing products and of a commercially available DBMS. The results of this work are, in turn, being incorporated into partners' products on UNIX and VAX VMS.

Since much of the ASPECT project is concentrated on the infrastructure, we look to other sources for many of the tools which will run on ASPECT. Some of these tools will be produced by the industrial partners, but ASPECT is also a potential base for tools developed in other Alvey projects and perhaps also other projects like ESPRIT's SPMMS. These tools will use ASPECT most effectively if they exploit the Public Tool Interface, and to aid this ASPECT will produce a formal definition of its PTI using the notation Z, developed at Oxford. Far more tools, of course, have been and will be written simply for UNIX, and ASPECT will integrate these tools through its Open Tool Interface.

SUMMARY

Although ASPECT cannot, of course, address all the problems of software development, it does address the major IPSE requirements and is a prototype of the next generation of IPSEs. In particular it supports the whole lifecycle by providing a sound framework for tool integration; it can be tailored to any method or collection of methods; it provides computing power and a highly functional interface at the disposal of the user and, by its physical distribution and support for controlled sharing it is a powerful environment for development in the large.

* UNIX is a trade mark of AT & T Bell Laboratories.

Three experimental multimedia workstations -a realistic utopia for the office of tomorrow-

Helmut Balzert
Research Department
TRIUMPH-ADLER AG
Nuremberg, West Germany

Abstract

The office of the future needs different multimedia workstations for different user groups. The architecture and the highlights of our multimedia office environment are sketched. The manager workstation has a completely new human-computer-interface: a horizontal flat panel built into an office desk. On the flat panel a touch-sensitive foil is used for input. Virtual keyboards can be displayed on the flat panel if needed by the application. A pencil with a built-in ultrasonic-transmitter is used as a pointing and handwriting device. Our model of multimedia interaction and communication is presented. A detailed explanation of how the processing of office procedures is implemented on our experimental workstations is given.

1. Introduction

In the past, only specialists were able to operate a computer. Generally, a long training phase was necessary: the human had to adapt to the computer. Now software and hardware technology is ready to change the situation completely: The computer is able to adapt to the human.

This ability is a necessary prerequisite for the office of the future. The acceptance of new office systems depends on the following conditions:

- Very short training phases
- Only little change in the current working style or evolutionary change
- Consistent and uniform interface design
- Direct manipulation via alternative multimedia communication channels
- Additional comfort

In the next chapter we will explain our human-computer-communication concept. Some facts about office activities are summarized in chapter 3. The architecture of our multimedia office environment and the hardware highlights are described in the subsequent chapter. Some important office scenarios including our software highlights are sketched in chapter 5. The last chapter contains a resumee and perspectives on the future.

2. Human-computer-communication

In human-computer-communication two forms of communication can be distinguished: explicit and implicit communication /Fisc 82/.

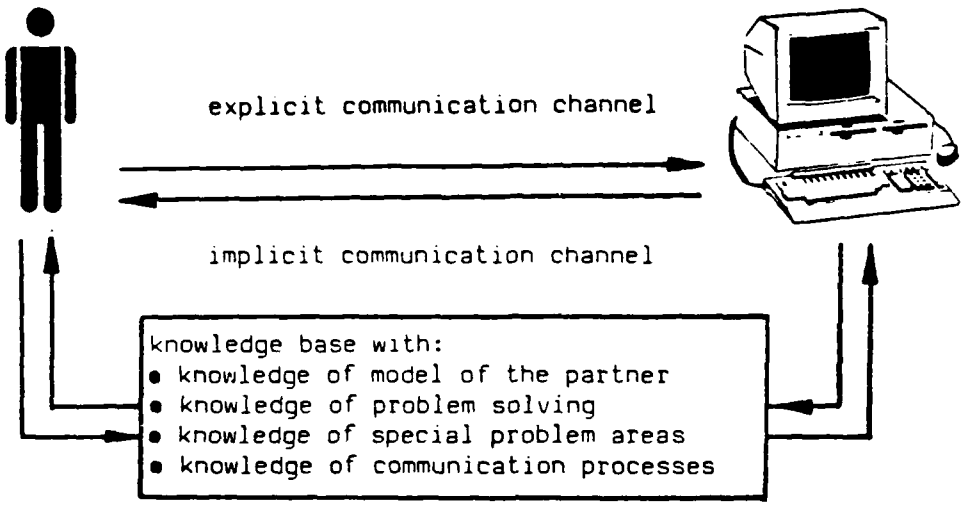


Fig. 1: Human-computer-communication model

In order to obtain optimal human-computer-communication the explicit and the implicit communication-channel must be very broad.

This article concentrates on the explicit communication channel. Today the explicit communication channel is narrow:

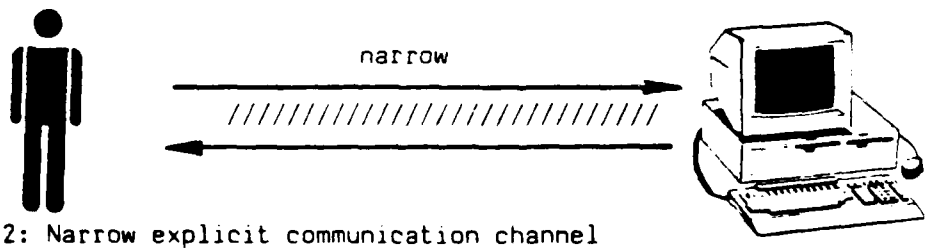


Fig. 2: Narrow explicit communication channel

Normally the communication is reduced to input via a keyboard and to output via a display. Modern systems like Xerox Star, Apple Lisa & Macintosh improve the communication using full graphic displays with icons and a mouse as a pointing device.

One way to improve the acceptance of new office systems is the use of a broad multimedia explicit communication channel:

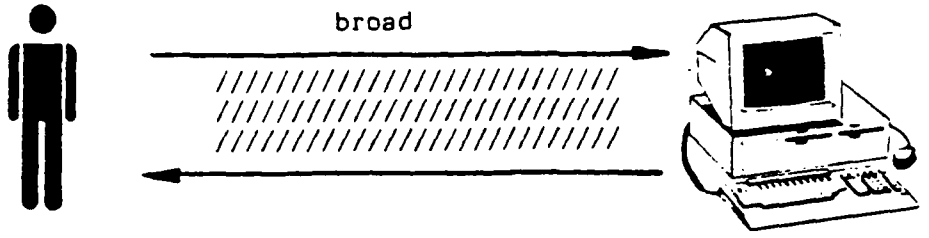


Fig.3: Broad explicit communication channel

Todays software & hardware-technology is ready to realize such a broad communication channel. This basically means that the user has different communication techniques availabe for the input of commands and information.

3. Office activities

Workstations in offices will, in addition to common applications, support various office activities that are not specific to particular types of workplaces. Examples of such activities are computerized dialing, filtering information from incoming mail, scheduling meetings etc. (see, e.g. /ElNu 80/).

The differences between different workplaces in offices should not be neglected, however. A careful inspection of present-day office work reveals typical differences. Searching, e.g., is more important for knowledge workers than it is for managers. On the other hand, managers will spend more time dictating than clerks or knowledge workers do. Writing will remain a typical activity for secretaries.

Modern information technology certainly will change the office, not only as far as the technology installed is concerned. There will also definitely be a high impact on work structure and work distribution. It would be going to far to address these questions here. It shall be noted, however, that a trend to more flexible structures, to an integration of different functions at a single workplace seems to emerge.

This does not mean that, some day in the future, all office workplaces will look alike. Differences in tasks and attitudes will remain. The three different prototype workplaces that have been developed at TRIUMPH-ADLER's basic research reflect this and have been tailored to three different types of office workplaces:

- a manager's workplace
- a secretary's workplace
- a knowledge worker/clerk's workplace

Developing such workplaces and testing them in real office environments will help to determine user needs and drawbacks that can not be foreseen from a technological viewpoint. It enables

developers to arrive at solutions to office problems and not just present sophisticated pieces of technology.

4. Architecture of a multimedia office environment

Fig. 4 presents the architectural concept of our environment.

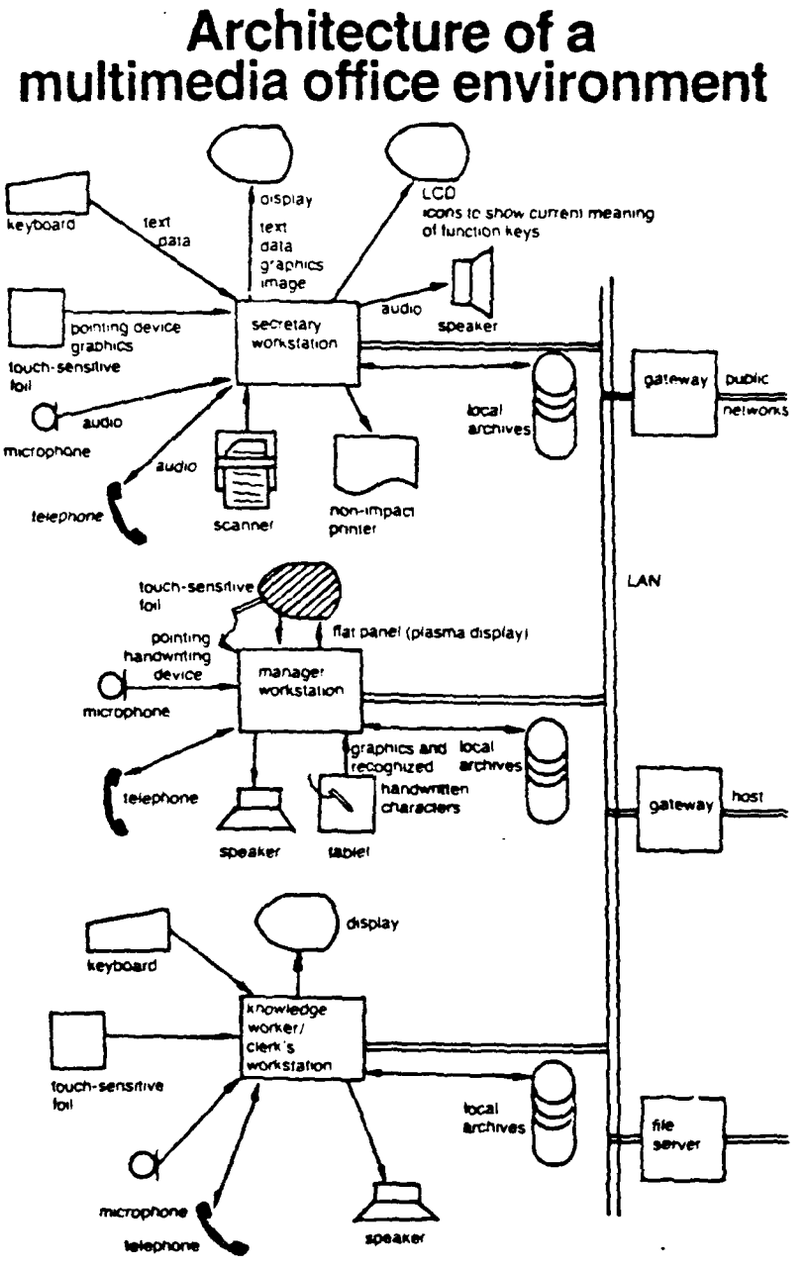


Fig. 4: Multimedia office environment architecture

Fig. 5, 6 and 7 give a visual impression of our existing workstations.

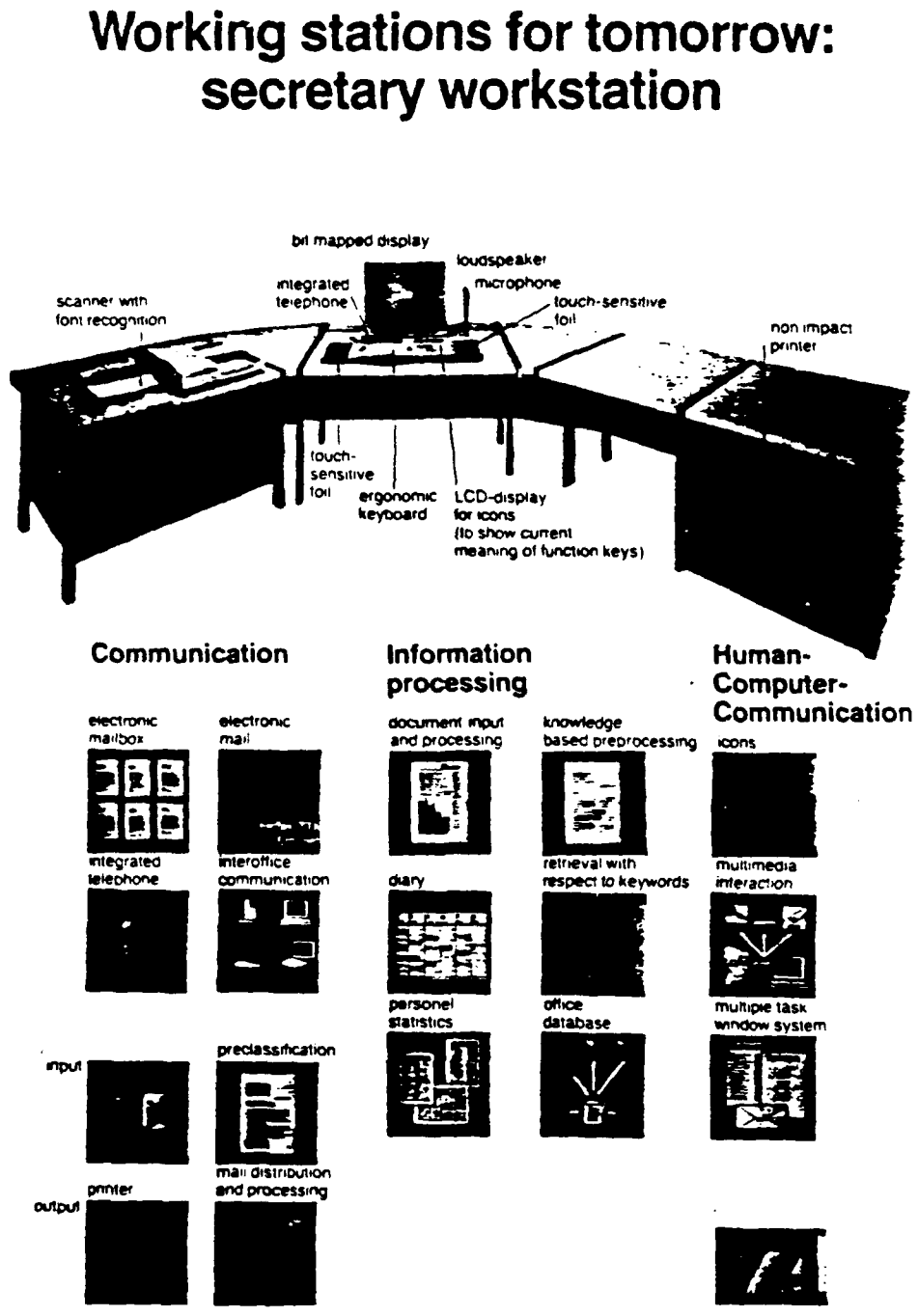


Fig. 5: The experimental secretary workstation

Working stations for tomorrow: manager workstation

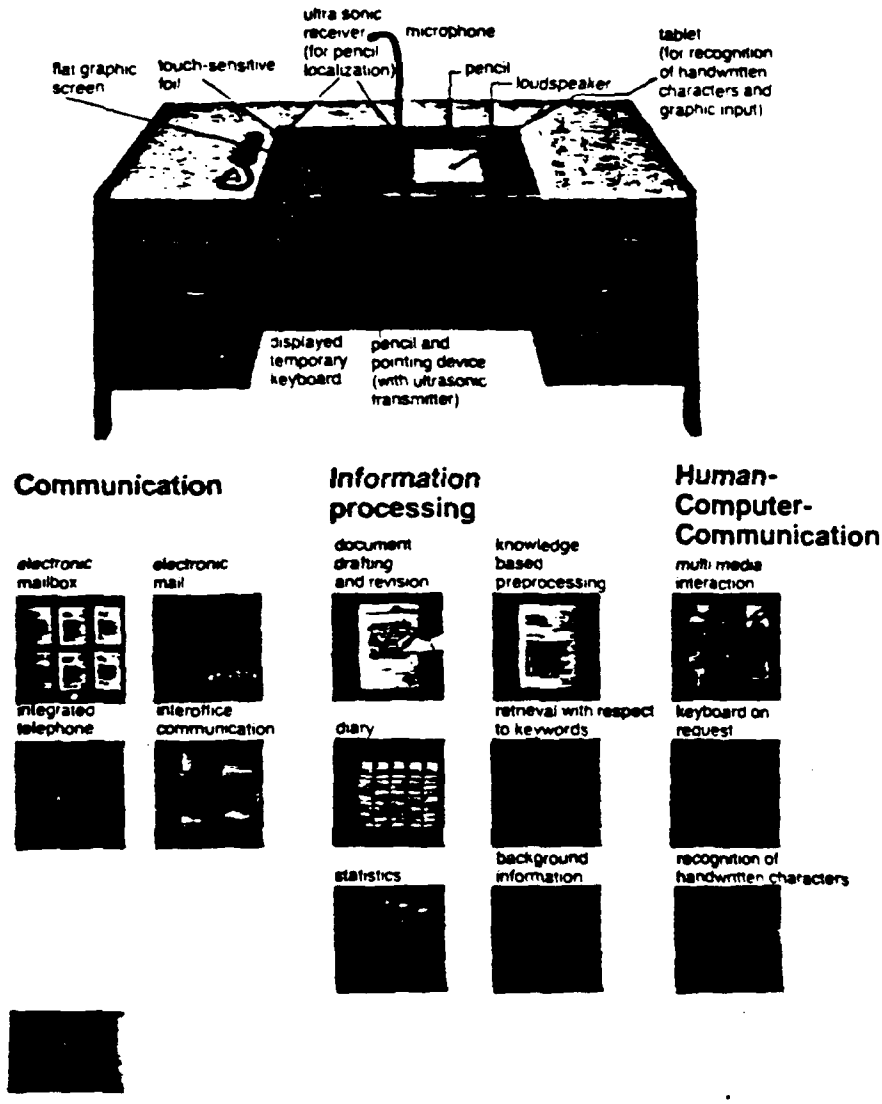


Fig. 6: The experimental manager workstation

Working stations for tomorrow: knowledge worker/clerk's workstation

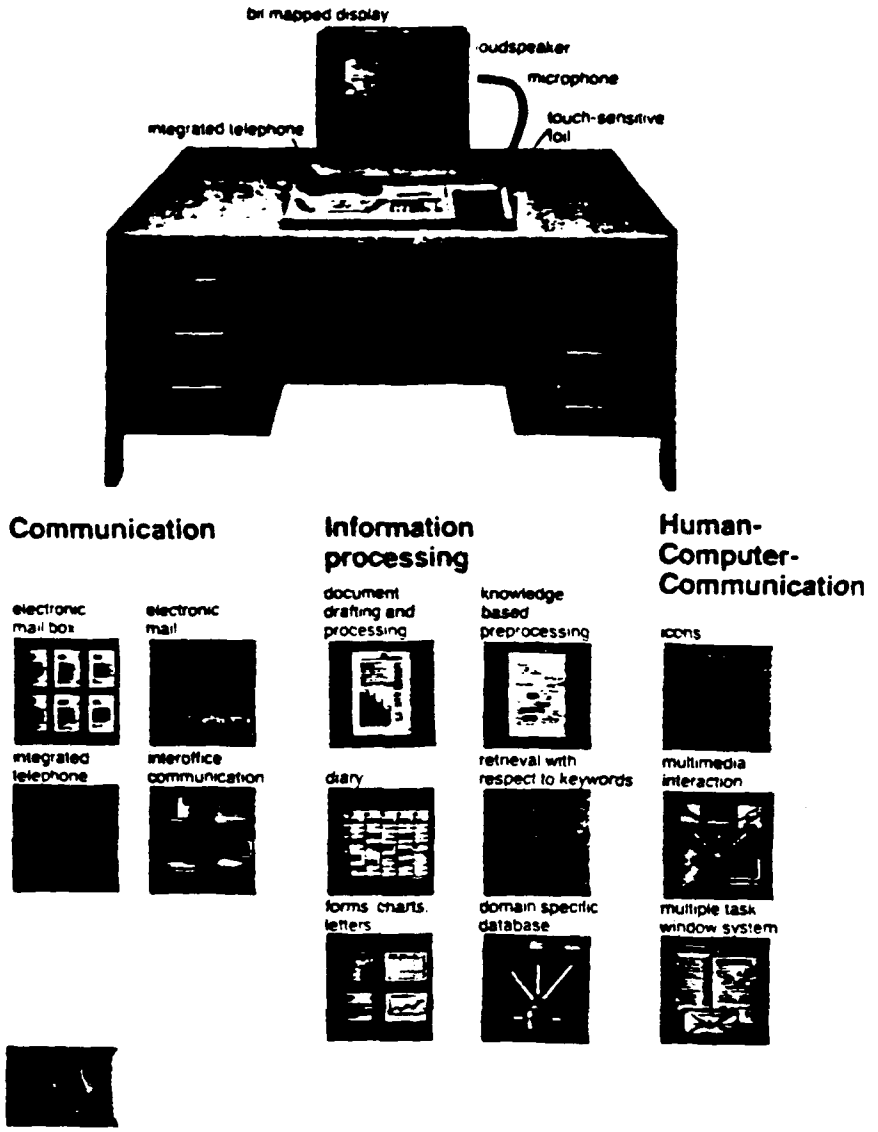


Fig. 7: The experimental knowledge worker / clerk workstation

The hardware highlights of the three workstations:

The secretary workstation:

An ergonomic keyboard with two separated blocks of keys is used

for text input. A LCD-display was placed above the free programmable function keys. It displays the current meaning of the function keys, using icons.

A touch-sensitive foil is fixed to the left and to the right of the keyboard. Each touch-sensitive foil is used as a pointing device like a "static mouse". We want ascertain if it is useful to have a pointing device both on the left and on the right side. It is possible to use the foils with different scales: a movement across the foil implies a movement of the cursor across the whole screen with one of the foils. Using the other foil. only a smaller part of the screen is passed with the same movement. Thus, "global" movements of the cursor over longer distances are carried out easily as well as "local" movements, i.e. very fine and precise pointing operations. To some extent, this may be regarded as a zooming effect. Sustantial results are not available yet as our experiments have just started.

Another advantage of such a foil, in comparison with a mouse, is that it can be used much better as an input medium for graphics and handwriting.

Also integrated in the keyboard is the telephone.

The manager workstation:

Because a manager normally does not type a lot of text or information he does not need a traditional keyboard. In addition a lot of managers are not willing to use a keyboard.

We have therefore developed a completely new human-computer-interface: a horizontal flat panel (in our case a plasma display) is built into an office desk. Above the flat panel lies a tranparent, touch-sensitive foil, which can be used as an input device for the virtual keyboards that can be displayed on the flat panel, according to the application. The second input medium is a pencil with a built-in ultrasonic transmitter. In one mode it can be used as a pointing device. Each time the pencil is pressed on the screen, an ultrasonic impuls is transmitted. Two small microphones receive the impuls. A processor computes the position of the pencil with a tolerance of 1/10 mm.

In a stream mode, the transmitter sends more than 100 impulses per second. In this mode, it is possible to do handwriting with the pencil on the flat panel. The handwritten text or graphics will be echoed on the flat panel in real-time.

A graphic tablet in the manager desk allows the recognition of handwritten block letters. It also can be used as an input medium for graphics.

The knowledge worker / clerk workstation:

Analog to the other workstations, this workstation is equipped with an integrated telephone, a microphone and a loudspeaker. The keyboard is connected with a touch-sensitive foil as a pointing and graphics input device.

5. Scenarios of office procedures

Scenario 1: Processing of incomming paper mail

Even in the office of tomorrow not all communication partners will have an electronic mailbox. Therefore a part of the incoming mail will be paper mail.

In our office architecture, incoming paper mail will not go further than the secretariate and / or the mail department.

A scanner transforms paper mail into electronic mail. Today we use a text scanner which recognizes type-written characters and can differentiate between different type sets. In future, we expect scanners which can process mixed modes: graphics will be transfered bit by bit, type-written text and printed text will be recognized.
This transformed paper mail will be processed by an expert system. It tries to locate the sender, the addressee, the date, the subject and the type-written signature, if it exists.

If the sender and the addressee can be recognized in the letter, the expert system then searches automatically in the archives, belonging either to workplace, the department or to the whole company, to verify whether the addressee and the sender are known.

If yes, the expert system sends the letter via electronic mail in the addressees mailbox. The secretary will not even notice this process, let alone play a part in it.

In the other case, the secretary gets the letters and documents in her mailbox to do a manual preprocessing.

Incoming electronic mail which is not transfered directly to the mail box of the target receiver, is treated in the same way.

Scenario 2: Preclassification, filtering and presentation of incoming mail

Before incoming mail reaches the mailbox of the office worker it will be preclassified by an expert system. If the mail did not come via the secretary's workstation to the mailbox, then a preprocessing analog scenario 1 takes place first.

The expert system searches in the personal archives of the office worker to look up all key words which the receiver uses to

classify his letters and documents. Then these key words will be looked for in the incoming mail and will be marked (through inverse video).

Each office worker may give priorityes to keywords and may also restrict these priorities to a certain duration or specific time intervals. In case the incoming documents contain one or more of these priority keywords and, if they exist, obeys the corresponding time restriction, this document will be displayed in a special representation form like inverse video, e.g., or is announced by a loudspeaker, etc.

The opposite is also possible: to determine senders or organizational units whose mail will be rejected. This mail will be sent back to the sender automatically or will be deleted.

This feature is one means of coping with the increasing flood of incoming information that is likely to be a result of widespread electronic information exchange. It will become very easy to send out more and more mail or electronic copies to more and more people. Our aproach represents a step towards a flexible and individual solution.

The implementation of our developed expert system is described in /Woehl 84/.

Scenario 3: Processing of mail

If an office worker looks into his mail box, the mail will normally be displayed in form of so-called miniatures. Miniatures are document icons and show the document considerably reduced in size. Its line and paragraph structure are visible, but not legible, the user gets a small visual image of his mail (fig. 8).

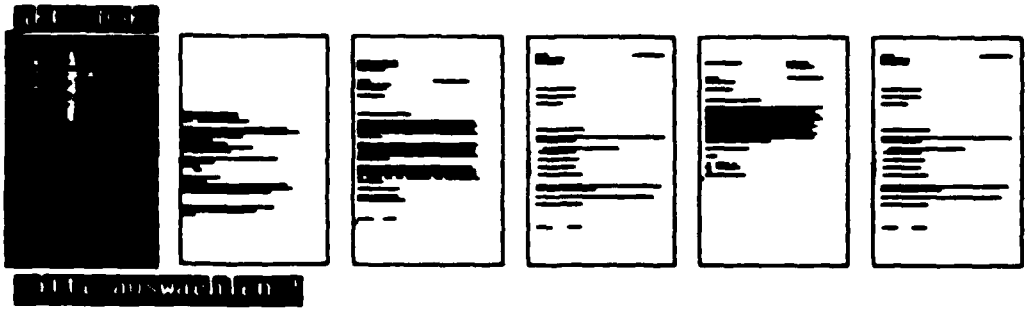


Fig. 8: Document miniatures

This approach helps in choosing documents, since it takes advantage of the excellent visual recognition capabilities of humans. Of course, the user is free to change the presentation of the mail to his needs and personal wishes.

The user chooses a document from his mail that he wants to read by pointing to it. The complete document is then displayed on the

screen. The first thing he might do is change the keywords chosen by the pre-classification system. He can then file it, send it to somebody else or start with appropriate tasks. He might, e.g., put some remarks on the letter using hand-writing or speech and design an answer, send the letter and the concept of the answer to a colleague or a senior executive.

Scenario 4: Remote access to mail

One of the problems which business people who travel frequently face is the access to their mail. It is not too difficult to develop systems that allow one to listen to speech mail that has come in during the day in the evening via the telephone. In addition to this, the technology of full-synthesis speech will provide the same opportunities for typed mail. The system will read these messages aloud. Of course, the user will be able to give comments on the various messages for his staff.

Scenario 5: Writing and sending letters

Designing a letter traditionally is done on paper or using a dictation machine. New technology allows for new possibilities. Handwriting still is possible. It is not done on paper, however, it takes place directly on the screen, using techniques as described above. This manuscript is then transferred to the secretary electronically. It can be seen in one window of the screen, while the secretary prepares the typed version in a second window. The typed letter is then sent back and signed by the corresponding person, who, of course, again uses handwriting. The letter then is ready for mailing. If the receiver can be reached electronically, this means is used. Otherwise, the text is printed out on a non-impact printer and mailed.

Letters for inhouse-use that do not have to be typed can be sent even more conveniently. The writer of the letter specifies the intended receiver and, if necessary, his address in block letters. Thus both the text of the message and the intended adresss are handwritten. The underlying system is able to recognize the information on the adressee. A mere push on a "mail" key then is enough to transmit the mail to the intended receiver.

Scenario 6: Using the telephone

When using a telephone, people are interested in talking to people, not in dialing numbers. In modern office systems, the system will keep the list of telephone numbers and will also do the dialing. The user only has to specify to whom he wants to talk and let the system do the rest. According to the principle of multimedia communication he can do this in various ways: type the name of the person, point to the name in the telephone list or just tell it to the system via the microphone. The user will not have to pick up the receiver during the dialing process. When the person called picks up the receiver, his voice is heard

through a loudspeaker, and this is when the person calling picks up his receiver.

6. Conclusions and outlook

- Tomorrows office will be multimedia: both multimedia documents and multimedia human-computer-communication
- Systems will be more flexible and adapt to specific user needs and habits
- The user will be freed from the burden of knowing locations, numbers, etc. he says what he wants - and lets the system take care of how to do the job
- Knowledge-based systems will extend the use of modern technology from supporting routine-tasks to supporting more complex and sophisticated tasks
- With the integration of features like handwriting in electronic systems, office systems of tomorrow will be able to cope much better with office tasks, procedures and habits - thus leading from partial to complete solutions
- Modern desks will on the outside look more and more like those of the pre-electronic age. Horizontal displays will make traditional terminals disappear. The modern office desk will look similar to the desk of the 19th century clerk. The electronic inside, however, provides help and services the latter could not even have dreamt of.

Acknowledgements

I would like to express my appreciation the members of the research department of TRIUMPH-ADLER for their participation in the work described in this paper. I am particularly indebted to my colleagues A.Fauser and R.Lutze for their contributions and discussions.

Literature

/EGLT 78/ Engel/Grappusz/Lowenstein/Traub
An Office Communication System
IBM Systems Journal 18 (1979), pp.402-431

/ElNu 80/ Ellis, C./Nutt, G.
ACM Computing Surveys 12 (1980), p.27-60

/Fisc 82/ Fischer, G.
Mensch-Maschine-Kommunikation (MMK): Theorien und Systeme
Habilitationsschrift Universitaet Stuttgart, 1983

/MSIS 83/ Mokawa/Sakamura/Ishikava/Shimizu
Multimedia Machine

IAP 83, pp.71-77

/OtPe 83/ Otway, H./Peltu, M. (eds.)
New Office Technology. Human and Organizational Aspects
Frances Pinter Publ., London

/UFB 79/ Uhlig/Farber/Bair
The Office of the Future
North-Holland

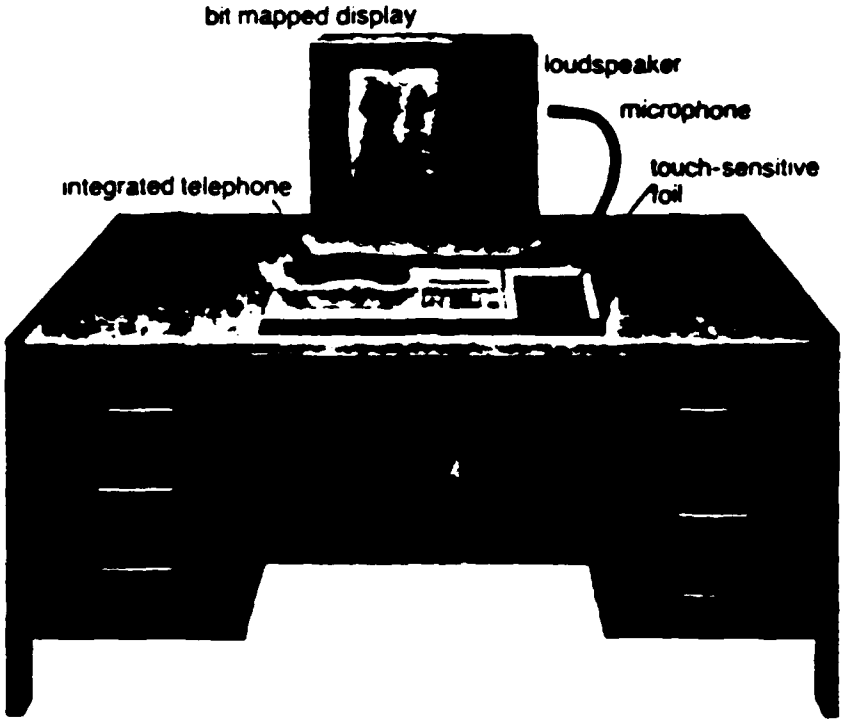
/Woehl 84/ Automatic classification of Office Documents By Coupling Relational Data Bases and PROLOG Expert Systems
Proc. 2nd Conference on VLDB, Singapore, 1984

Adress:

Dr. Helmut Balzert
TRIUMPH-ADLER AG
Nuernberger Str. 159

D-8510 Fuerth
West-Germany

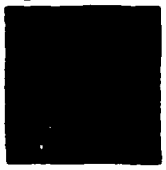
Working stations for tomorrow: knowledge worker/clerk's workstation



Communication

electronic mail box

electronic mail



integrated telephone

interoffice communication

Information processing

document drafting and processing

knowledge based preprocessing

diary

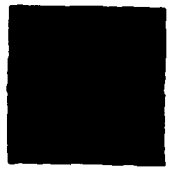
retneval with respect to keywords

forms, charts, letters

domain specific database

Human-Computer-Communication

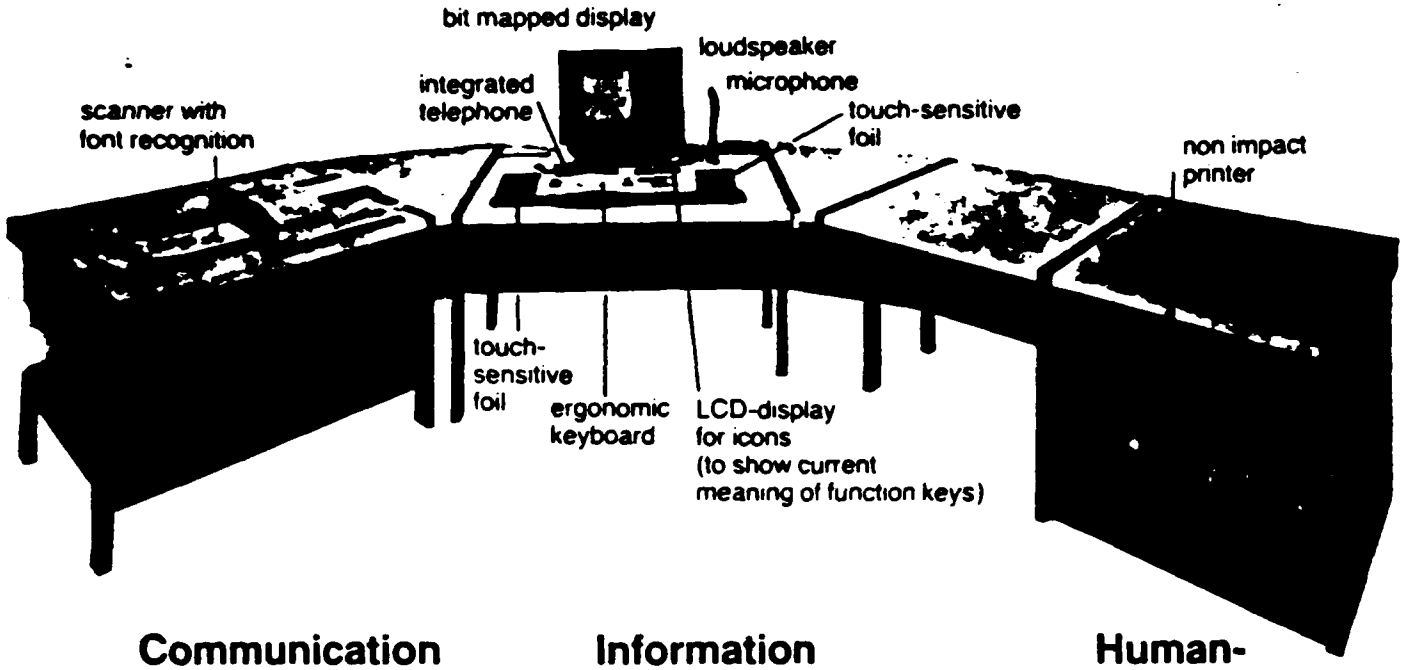
icons



multimedia interaction

multiple task window system

Working stations for tomorrow: secretary workstation



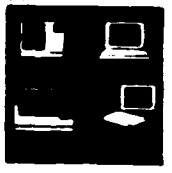
Communication

electronic mailbox

electronic mail

integrated telephone

interoffice communication



input

preclassification

mail distribution and processing

output

printer

Information processing

document input and processing

knowledge based preprocessing

diary

retrieval with respect to keywords

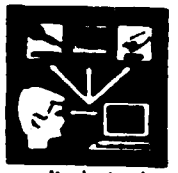
personel statistics

office database

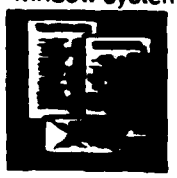
Human-Computer-Communication

icons

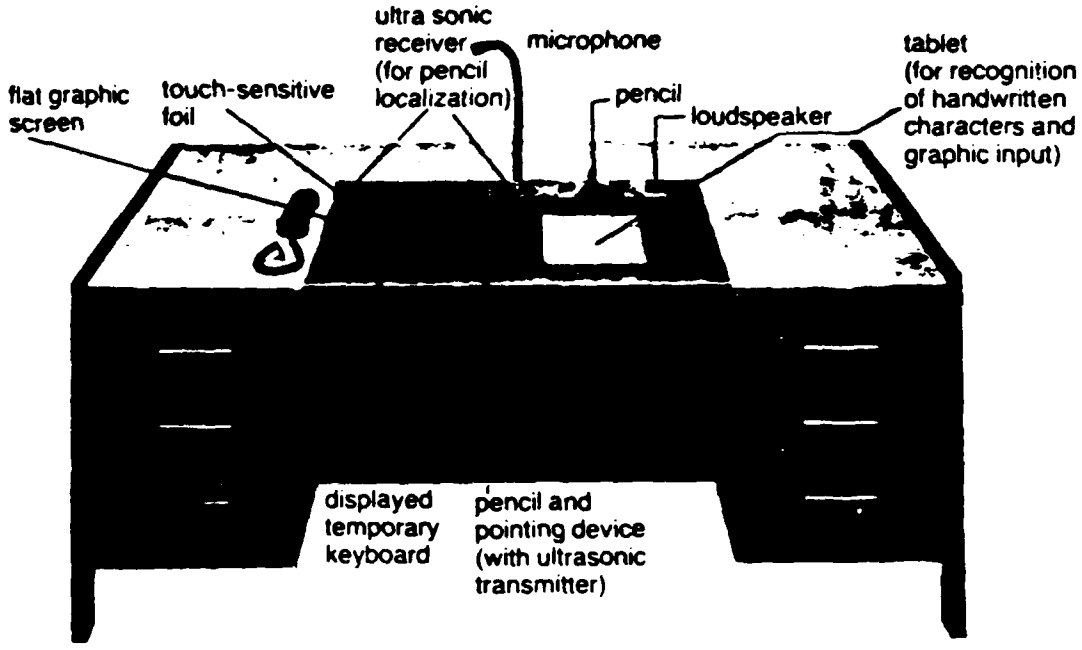
multimedia interaction



multiple task window system

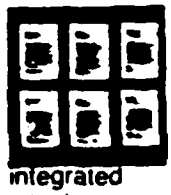


Working stations for tomorrow: manager workstation



Communication

electronic *mailbox*



electronic *mail*

integrated telephone

interoffice communication

Information processing

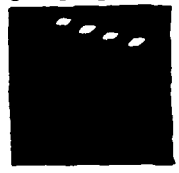
document drafting and revision

knowledge based preprocessing

diary

retrieval with respect to keywords

statistics



background *information*

Human-Computer-Communication

multi media interaction

keyboard on request

recognition of *handwritten characters*

FORMAL METHODS: PRESENT AND FUTURE

*M I Jackson

Praxis Systems plc
20 Manvers Street
Bath BA1 1PX
United Kingdom

1 Introduction

It is widely accepted that industrial software production has a number of associated difficulties and problems. These include the following:

The proportion of system costs due to software has increased dramatically over the last 20 years and is continuing to do so. This poses major problems for technical management who are frequently qualified and experienced in more traditional engineering disciplines.

A major skill shortage exists in software and is expected to continue to increase, (for example, it has been estimated by the US DoD that the USA is currently short of 200,000 programmers and will be short of 1 million by 1990).

The quality of delivered software is frequently inadequate in performance and reliability. For example, a US army study showed that for a 2 million dollar budget, less than 2% of the software was used as delivered.

Software projects are subject to frequent cost and timescale over-runs. The causes are usually inadequate engineering methods to handle the complexity of systems and inadequate project management techniques.

The discipline of software engineering is in its infancy. There is at present relatively little underlying theory or generally accepted good practice on which to base an engineering approach. As a consequence, software systems tend to be 'crafted' rather than engineered.

These (and related) problems have by now received world-wide recognition. A number of major initiatives have consequently been instituted to address these issues, for example, the Alvey Software Engineering programme and the ESPRIT Software Technology programme.

*previously with: Standard Telecommunication Laboratories Limited
London Road, Harlow, Essex CM17 9NA, United Kingdom

SXEAAX

From an industrial point of view, it is clear that the following global objectives should be addressed:

Increased productivity in software production (to address skill shortages)

Higher quality in software products, eg improved reliability, performance and adaptability.

More effective use of human and machine resources, eg by automating routine tasks as far as possible thus allowing highly skilled and scarce staff to concentrate on intellectual tasks.

These objectives need to be addressed by developments in a number of areas. The remainder of this paper will concentrate on one area in particular - formal methods.

2 Formal Methods

The introduction of formal methods of system development is seen as a major step towards improving the software engineering process. Formal methods are rigorous engineering practices based on mathematically formal foundations. Unlike most of the present day approaches in widespread use, formal methods provide the means of expressing system designs concisely and unambiguously and in such a way that the behavioural aspects of systems can be explored through formal logical manipulation.

The Alvey Software Engineering Directorate has established a major programme in the formal methods area (ALV84). This programme emphasises three areas of activity: the rapid exploitation of mature formal methods, the industrialisation of promising methods so that they can be exploited in the near future, and fundamental research to provide more powerful methods in the longer term.

The Alvey formal methods programme foresees a significant number of benefits arising from the introduction of formal methods. Most significantly, they are seen as providing the scientific basis underlying software construction. They will allow at least the level of confidence in software designs as in other more established branches of engineering, and will be the key to the certification of software in safety-critical applications. Formal specifications are seen as particularly important for providing a firm contractual basis for the interaction between the supplier of, and the client for, a piece of software. They also provide a natural basis for rigorous interface descriptions thus simplifying the problem of reusing software components. Finally, the use of formal methods early in the life cycle should uncover many specification and design errors which otherwise might not have been detected until system test and operation when their repair would be very (if not prohibitively) expensive.

The Alvey programme characterises a mature method as having the following attributes:

> Books, technical reports and journal articles are widely available and accessible to all sections of the community.
>
> Training programmes have been developed and given field trials in industrial contexts.
>
> Industrial case studies have been conducted and the results published. Evaluations of these case studies have also been conducted and published.
>
> The method has had some, perhaps small scale, production experience in industry.
>
> Some useful tool support, possibly of an experimental nature, should exist.
>
> The major strengths and weaknesses of the method are reasonably understood.

These characteristics are rather demanding, but they are the essential minimum that must be achieved before a method can be successfully introduced into an industrial environment. They indicate the significant level of effort that must be devoted to 'industrialising' promising ideas from the research community before they can be assimilated by industry. Currently very little effort is applied to this task, which is one of the major reasons for the gulf between theory and practice. Even if a method is deemed mature according to the criteria given above, much more experience will be required before it achieves widespread adoption in industry.

3 STC Experience of VDM

Standard Telecommunication Laboratories, the central research laboratory of STC, has been conducting research into formal approaches to system development since 1979. In 1982, a major new project in the office systems area requested advice on formal specification methods. Of the methods under consideration at the time, one in particular, the Vienna Development Method (VDM) was recommended for consideration.

VDM originated in the Vienna Research Laboratories of IBM and is most closely associated with D Bjorner and C B Jones. The reasons for its recommendation were largely pragmatic, for example:

> Considerable user experience of the method already existed within organisations such as IBM and the Danish Datamatik Center. A number of case studies have been published, for example in the book by Bjorner and Jones [BJ082]. The strengths and weaknesses of the method are well understood.

Training and consultancy in the method were available. Professor C B Jones, now of Manchester University, was willing to present an established 2 week course to the project team and to provide continual support through consultancy.

A well-written and accessible text book produced by Professor Jones was available to support the course material.

In the summer of 1982 an evaluation exercise was conducted in order to assess the suitability of VDM as a specification vehicle for the proposed project. This exercise was organised as follows:

A subsystem of the project was chosen as the case study to be used.

A small group of analysts, having no previous familiarity with formal methods, were selected and introduced to VDM.

Two STL staff members were asked to support the analysts as consultants (alongside Professor Jones).

An observer, independent of the analysts and consultants, was appointed to identify assessment criteria before commencement of the evaluation, to observe the conduct of the exercise, to write the evaluation report and to produce recommendations to management.

The chief system designer was involved from time to time to answer questions regarding the requirements and to play the role of 'customer'.

The evaluation exercise took as its input an English language statement of requirements. Its outputs were the corresponding VDM specification, the evaluation report and recommendations for VDM.

In the light of this exercise, STC decided to adopt VDM for the project, and set in hand work in the areas of training, standards and support tools. The method was subsequently adopted for other internal projects.

The experience of VDM can be summarised as follows:

The identification of abstract data structures in the formal specification aids conceptualisation of the eventual product and supports the optimal choice of operations and functions for the eventual users of the system. The iterative evaluation of the specification against user requirements assists dialogue with the customer and continues until a version acceptable to the customer is produced.

The VDM specification language provides a number of useful thinking tools which allow the analysts to distinguish more easily between WHAT the system would do and HOW it should do it. The analyst is also forced to consider many more aspects of the requirements than with previous (non-formal) analysis techniques.

The precision of the specification language reveals many previously unrealised anomalies and inadequacies in the informal statement of requirements. Rigorous reasoning about the specification improves confidence in its adequacy. An improved natural language requirements specification can be produced based upon the formal specification.

As a result of growing interest in VDM within the company, a programme of training courses was developed. These comprise:

A Perspective on VDM. This is a 3 day course aimed at project managers, team leaders, consultants and support staff from areas such as quality assurance, technical documentation or marketing and who require a sound understanding of VDM. It provides a broad view of formal methods, the ability to read and review VDM specifications, and advice on the introduction of VDM into a project.

VDM Foundation Course. This is a 10-day intensive course for technical staff wishing to produce system specifications written in VDM and for those needing to design systems to satisfy VDM specifications. Students completing the course should be able to read and write VDM specifications, demonstrate that a specification meets a requirement and demonstrate that a design meets a specification.

These courses are now offered to the public by STC which operates a commercial software and training service.

Various language development activities for VDM have also been conducted in STC. In the short term, it was recognised that there was an immediate need for facilities to allow the specification process to be carried out 'in the large' and for designs to be documented alongside the specifications that they implement. STC consequently undertook the design of an extended design language which included a module construct and a design pseudocode besides the essential components of the VDM specification language. (This work is heavily influenced by work undertaken at IBM's Boblingen laboratory on the SLAN-4 language [BEI83]. This language is supported by a UNIX-based syntax checker.

In the longer term, it was recognised that in order to extend the VDM specification language in a coherent way, and in order to build more powerful automated support tools, it would be necessary to improve the standard of definition of the existing specification language. Consequently, a joint activity was started by STL and Manchester University to define for STC a VDM Reference Language to be used as the STC standard. To date, documents describing the concrete syntax, abstract syntax, type model and context conditions of the Reference language have been issed and further work on semantics is underway. In addition, it is intended to develop a more extensive support toolset around the Reference language, and a UK Alvey project has recently commenced for this purpose. Recently, work has begun to develop a UK Standard Definition of the Reference Language based on the work carried out in STC.

In conclusion then, the STC experience indicates that, as a method for formal specification, VDM can be successfully introduced, if appropriate levels of investment are made in the areas of:

Evaluation, by case studies, of the suitability of the method for the application area of interest. The evaluation criteria should be properly defined, monitored and assessed.

Training for staff, of a professional quality.

Consultancy, as appropriate to particular needs, by skilled and experienced personnel.

General support, for example, by developing and implementing corporate standards.

In the longer term, support tools, such as a specification-oriented database and a syntax-directed structure editor.

For more information on VDM, readers are referred to the book by C B Jones [JON80]. Readers requiring further details of STC's experiences with VDM are referred to a recent paper [JAC85].

4 Formal Methods in the Life Cycle

There are many ways of viewing the software life cycle, the best-known being the standard 'Waterfall' method. In the alternative 'Contractual' view of the life cycle, each phase is regarded as involving two parties, one fulfilling a 'customer' role, the other fulfilling a 'supplier' role. So, for example, at the requirements specification phase, the customer role will be filled by the real customer for the system, while the supplier will be the analyst whose function is to supply the formal specification. The customer will provide an initial informally worded requirements statement and the analyst will produce a first attempt at a formal specification which he first verifies for internal consistency and completeness. (In effect he asks (Is this a specification of some system even if it is not quite what the customer wants?').

Having produced this first attempt, he must then demonstrate the effects of the specification to the customer. It is obviously unreasonable to expect the average client to be able to read the formal notation directly, though there are classes of (more sophisticated) customer where this does not apply. In general the analyst will use a process known as 'animation' to demonstrate the effects and consequences of the specification. This might involve a symbolic execution technique, for example, by moving tokens around a Petri net, or might involve the interpretation of behaviour derivable by theorems from the formal specification in the client's domain of knowledge. Inevitably, the first specification will not match the customer's precise requirement so further iterations of this process will be necessary. Eventually, a specification will be produced to which the customer agrees and this can be forwarded to the design stage (of course, problems may be discovered later which cause the requirements to be reconsidered).

In the design phase, the analyst becomes the 'customer' and the designer 'supplier' with the responsibility for producing a design which is consistent with the specification. Once again, the processes of verification and validation are iterated until an acceptable design is produced.

These contractual relationships can continue through however many design and implementation phases as are necessary.

The contractual model presents a somewhat idealised view of software development, but is useful as a basis for examining how formal methods might be used. There is general agreement that formal methods should be used initially at the specification phase since the greatest cost savings will be made by locating specification errors early. At the design and implementation phases, the degree of formality used would probably be less, since, except in the case of very critical projects, the costs would probably outweigh the benefits. In most cases, a rigorous approach to design would be more acceptable, with the verification and validation techniques being more informal and traditional in flavour. Testing strategy could, perhaps, be based upon the formal specification. Post-developmental phases, such as maintenance or enhancement, would be considerably assisted if formal specification were kept in the project database since the implications of proposed changes could be more easily assessed.

Training and education are areas which require attention. It must be emphasised that the introduction of more formal methodologies *does not* reduce the skills needed to develop software. On the contrary, the newer methods require a different set of skills to traditional programming activities. Higher standards of education and professionalism are required. Courses must be developed (some already exist for the established methods) to provide analysts and designers with the necessary skills in particular methods. Managers, too, need training to manage reviews and monitor progress of projects using formal methodologies. Educational courses must be provided to give the necessary background in discrete mathematics which many experienced practitioners lack (though the relevant material is gradually being included in many undergraduate computer science courses).

5 Automated Support for Formal Methods

The most basic type of support tool is one that supports syntax and type checking. For any well-defined specification language a syntax and type checker can often be provided using standard parser generators. Coupled with a reasonable file or database system, such a tool would provide an invaluable aid to the system specifier and should be relatively cheap to provide. Other syntax-oriented tools include syntax-directed editors, pretty printers etc.

A more sohpisticated type of tool would support symbolic execution of specifications and designs. The difficulty of building such tools depends to some extent on the formal method being supported, but clearly such an aid would be most useful as a validation and animation tool.

Validation would also be supported by theorem proving aids which support formal verification of specifications and transformations during the various phases of design. Recently, there has been a trend away from large theorem provers which consume large amounts of computing resources to interactive systems which assist the human in carrying out a proof. Related is the work on automated program transformation which promises to assist the designer in producing correct implementations with regards to the specification.

The underlying support environment cannot be ignored. Developments such as the Ada APSE or the European PCTE promise much as the central database will be provided. If a proper database were provided, it should be possible to build libraries of reusable software components each having a formal specification. By composing specifications of existing parts, and by integrating them with the specifications of new parts, it should be possible to demonstrate that a larger system specification could be met. Thus, system development might become a more bottom-up activity with greater reuse of previous work than is possible at present.

Prototype versions of most of the tools mentioned above are already available, at least in research environments. It is reasonable to expect that production quality versions will appear over the next few years.

6 Conclusions

The next few years will witness an increasing use of formal development methodologies as users become more sophisticated and attempt to detect errors in the specification and design phases of the life cycle. Formal methods are already worthy of serious consideration as a specification aid and a rigorous design framework has also been shown to be highly practicable. The results of major research projects established under programmes such as Alvey and ESPRIT, will lead to more extensive and adaptable methods for future use, with a greater level of automated support.

Readers wishing to obtain a more detailed view of the current state of various approaches to formal methods are referred to the comprehensive survey by Cohen et al [COH 86].

7 References

[ALV84] Alvey Programme Software Engineering
Programme for Formal Methods in System Development
Alvey Directorate, April 1984

[BE183] Beichter, F, Herzog, O, Petzsch, H
SLAN-4: A Language for the Specification and Design of Large Software Systems
IBM Journal of Research and Development, Vol 27, No 6, November 1983

[BJO82] Bjorner, D, Jones, C B
Formal Specification and Software Development
Prentice Hall, 1982

[COH 86] Cohen B, Harwood, W T, Jackson, M I
The Specification of Complex Systems
Addison Wesley, 1986

[JAC85] Jackson, M I, Denvir, B T, Shaw, R C
Experience of Introducing the Vienna Development Method into an Industrial Organisation
Procs. Int. Conf. on Theory and Practice of Software Development (TAPSOFT), Berlin, March 1985. Published as Springer Verlaag LNCS 196

[JON80] Jones, C B
Software Development - a Rigorous Approach
Prentice Hall, 1980

A Retargetable Debugger

for the Karlsruhe Ada System

Author s): Peter Dencker
Hans-Stephan Jansohn

Version: 1.0
Date: 04 May 85
Source: dbgs:doc:redebug.max
Key:

---

*) Ada is a registered trademark of the U.S. Government (AJPO)

Table of Contents

1.0 Introduction

During the development phase of a program and - as well - during the maintenance phase it is very important that programming errors can be quickly located and removed from the program. Therefore a programming environment should very well support the analysis and location of errors. An appropriate means is an interactive debugging tool. The purpose of the debugging system described in this paper is to support the analysis of programs translated by the Karlsruhe Ada Compiler.

However, the Karlsruhe Ada Compiler is easy to retarget and easy to rehost. Several versions are already available (VAX/VMS, SIEMENS/BS2000, and SIEMENS/BS2000-MC68000 Cross compiler), several others are planned (Nixdorf 8890/VM/ESX, Perkin Elmer 3200/OS32, and VAX/VMS-LR1432 Cross compiler). Therefore it is very important that the debugging system is equally well retargetable and rehostable.

The scope of a certain version of the debugger are all programs (including tasking and real-time applications) translated by the corresponding version of the Karlsruhe Ada Compiler. This implies that for each version of the Karlsruhe Ada Compiler a corresponding version of the debugger is available.

The compiled Ada programs under test may either run in the environment of the host system, which contains the compiler, the program library, and the debugger or they may run on some target system (including embedded system) with more or less restricted communication channels to the host system.

The debugging system will be developed in two major stages:

(1) post mortem dump analyzer

(2) online debugger

The first stage, the post mortem dump analyzer, is partially implemented. The other will be implemented within the next year.

This paper is organized as follows. The concept of post mortem dump analyzing and online debugging are explained in the

next two Sections. In Chapter 2 the general requirements and capabilities of the debugger are listed to set up the debuggers functional range. In Chapter 3 and 4 we show a debugger design which suits the requirements and capabilities established in Chapter 2.

1.1 Post Mortem Dump Analyzer

The basic idea of a post mortem dump analyzer is that when the program stops with an error, in Ada e.g. when the program is abandoned because of an unhandled exception, the contents of memory is saved on file, the dump file. Then the post mortem dump analyzer examines this dump file and extracts information valuable for the programmer in order to locate the programming error. In this way, for instance, the values of variables and parameters can be inquired. To this end the post mortem dump analyzer needs information about the runtime organization and information from the compiler and the linker/loader which is stored in the program library.

1.2 Online Debugger

An online debugger has all capabilities of a post mortem dump analyzer. The difference is that the debugger communicates online with the program under test. If the program under test is interrupted control is transferred to the debugger. It then may examine the contents of the memory of the interrupted program in the same way as the post mortem analyzer examines the dump file. Afterwards execution of the program can be resumed.

Additionally the online debugger allows the programmer to change the state of the interrupted program (e.g. change the values of variables) and to control the execution of the program (e.g. by means of breakpoints).

2.0 Requirements on the Debugging System

Besides the global requirement of retargetability we impose several other general requirements on our debugging system.

(1) No recompilation must be necessary in order to run a program under control of the debugging system. This is important for transient errors in real-time programs.

(2) The ability to debug a program shall not impose overhead on the generated code. This means that the program must not be specially instrumented for debugging. This allows full debugging support even in a production environment.

(3) The executable code and data shall lie at the same memory locations regardless of whether the program is debugged or not. This is necessary to avoid errors which are due to placing the code at certain locations in memory.

(4) The debugger must provide a source level user interface. This means that all user interactions with the debugger are in terms of the Ada program. In this way, the programmer should have the impression that his program is run on an Ada machine.

(5) Access to low level information shall also be possible because Ada allows to interface the non Ada world through the pragma INTERFACE, through machine code insertions, and other low level features. Nevertheless for most users and in most situations this information is not needed to understand the state and execution of an Ada program.

(6) The debugger must be applicable in a host-target environment. It may e.g. run remote on the host system as a post mortem dump analyzer, or as an online debugger if an online communication channel exists between host and target, or directly on the target if the program library and sufficient resources are available on the target.

These general requirements are supplemented by the capabilities to analyze and interact with the program under test given in the next two Sections.

2.1 Analyzer Capabilities

(1) Inspection of the source code

(2) Inspection of the values of objects (variables, constants, parameters). The values are displayed in a form consistent with their representation in the source. For instance a record is displayed as a named aggregate. Access objects get a special format.

(3) Inspection of the status of currently existing tasks.
- name and type of task
- state (executing, suspended, completed,...)
- point of execution (in the source)
- tasks with outstanding calls on all entries of each task
- set of all open alternatives of select statements per task

(4) Inspection of the stack of subprogram activations for each task

(5) Navigation through the dynamic program structure (e.g. walk back, task inspection). In parallel the visible environment is changed.

2.2 Interaction Capabilities

(1) Placement of breakpoints at Ada statement and declaration boundaries

(2) Assignment of actions to breakpoints which shall be taken upon reaching a breakpoint

(3) Modification of data

(4) Interruptability of the program in execution by the user in order to inspect it at arbitrary moments

(5) Stepping through the program in granularity of declarations, statements, calls, or task interactions

(6) Tracing of executed declarations, statements, calls or raised exceptions

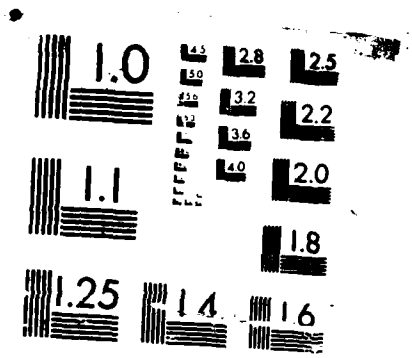
(7) Assignment of breakpoints to certain or a

AD-A155 434 PROCEEDINGS OF THE EUROPEAN SEMINAR ON INDUSTRIAL SOFTWARE ENGINEERING (2... HATFIELD POLYTECHNIC (ENGLAND). SCHOOL OF INFORMATION SCIENCE 3/3

UNCLASSIFIED H BALLERT ET AL. 10 MAY 85 AMD-5031-CE-02 F 5 12/5 NL

[redacted]

END
DATE FILMED
9 85



MICROCOPY RESOLUTION TEST CHART

exceptions

(8) Assignment of breakpoints to blocks under the condition that they are left by an exception.

(9) Hold (all) task(s), release task

In the following Chapters we show the design of a debugger for the Karlsruhe Ada System that satisfies these requirements.

3.0 Inspecting the Program State

Within this Chapter we assume that a program has been interrupted in some program state. We show how the debugging system can interpret this program state in order to answer the inquiries of the programmer.

Since we do not want to impose overhead on the execution of the program we do not collect information during its execution. Hence, the current program state is the only information the debugging system can access concerning the execution of the program.

The debugging system can, however, consider further information

(1) gathered by the compiler when compiling the individual compilation units of the program in the program library,

(2) from linking the program and

(3) about the runtime organization.

The runtime organization may very strongly depend on the target machine. However, these implementation details can be hidden behind a machine independent interface as long as we consider the same compiler.

The debugging system must access this information and derive from it a lot of data valuable for the programmer during the analysis of a particular error situation. The debugging system should work interactively to allow the inquiries of the

programmer to depend on the results of earlier ones.

In the following we discuss in detail how the debugging system can answer several characteristic questions concerning

(1) the currently executed source statement,

(2) the calling hierarchy and

(3) the values of objects.

3.1 Retrieving Source Statement Information

During debugging there is often a need to determine the position in the source that corresponds to a given code address or vice versa. For instance, the programmer may want to know which source statement raised an unexpected exception or he may want to interrupt the program when a certain position in the source is reached.

Since we consider a language with separate compilation, a position in the source is not uniquely defined by a source line number. Instead, we need a pair where one component indicates the compilation unit of the position. On the other side, since single statements may be distributed over several lines and since several statements can be written on one line, source line numbers are not appropriate for identifying the positions of single statements.

In the Karlsruhe Ada System the individual compilation units are stored in their Diana representations in the program library. The Ada source can be retained from the Diana representation of a compilation unit. Hence, it is suitable to use references into the Diana representations for identifying positions in the source.

For the following it not of interest how the positions in the source are identified. We assume only that they can be uniquely identified. We call such an identifier a source position.

A common technique for determining the source position of

the currently executed statement is to use a variable (in the runtime system) which always holds it. It is updated when the current statement changes.

This method has three great disadvantages:

(1) The code size as well as the execution time of the program are increased significantly (25-50%). This fact causes most programmers to switch off the generation of source statement information. So debugging of the programs is not further possible without recompilation.

(2) This method allows only to determine the position of the current executed statement. By this way, the name of the statement which e.g. called the subprogram containing the current statement cannot be obtained without storing it in the invocation frame. Further it is not possible to determine the code address(es) corresponding to a given position in the source (e.g. for implanting breakpoints).

(3) For rather complex statements more detailed information is needed for selecting the erroneous part of the statement. This would, however, result in much more overhead.

We propose another method which does not possess these disadvantages: The compiler builds tables which contain the mapping of code addresses to source positions and stores them in the program library. The debugging system interprets these tables for determining the position in the source which corresponds to a given code address. Since the relevant addresses are (normally) available during the execution of the program no additional code must be generated. For instance, the program counter contains an address corresponding to the currently executed statement, the return address is already stored in the invocation frame for subprogram calls. Hence there is no runtime overhead and the code size as well as the execution time is not increased.

Now we discuss how the tables containing the mapping of the code adresses to statement names are constructed. Since we consider a language with separate compilation facilities the compiler does only know the mapping of module relative code addresses to source positions. Therefore for each compilation unit one table is built containing the module relative

information.

When linking the program, the linker computes the absolute addresses of the code modules resulting from the individual compilation units. These adresses together with the module names are usually written on some file. We call it the linker listing.

Given a particular code address the debugging system looks into the linker listing and determines to which code module it belongs. In presence of code overlay additional information is required from the program state in order to resolve the resulting ambiguities. Then it computes the module relative code address and obtains the corresponding source position by inspecting the table built by the compiler for this module.

If a source position is given the code address(es) corresponding to this source position can obviously be computed in a similar manner.

We have implemented this method in the Karlsruhe Ada System. By a simple compactification method we could reduce the size of the compiler generated tables to about 20% of the size of the generated code. So the size of these tables does not cause any problems. It is even less than the increase in code size which would result from the other approach.

3.2 Determining the Dynamic Calling Hierarchy

In order to analyze an error situation the programmer has to inspect the source. The debugging system can support him by selecting those parts of the source which are associated in some way with the error situation. In the following we call a position in the source together with the context of the current program state a location. Starting from a location the programmer can inspect the static environment directly in the source.

However, this does not suffice. Additionally the programmer is interested in how the program came to a certain location. Especially, he wants to know the subprogram or entry calls which led to this location.

For Ada programs we therefore distinguish the following locations indicating the dynamic context of the interrupted program:

(1) Each currently existing task or the main program is interrupted at some location. This location is called the current location of the task or the main program.

(2) A location may lie within a subprogram. The location where this subprogram was called, is referred to as the calling location of it.

(3) A location may lie inside an accept statement. The location of the corresponding entry call is the current location of the task issuing the entry call. Hence, it suffices to know this task. We call it the calling task of it.

In order to be able to inspect each existing task the debugging system must know them all. Since in Ada each task depends on some master, all existing tasks can be enumerated if for each master all tasks which depend on it are known.

This leads to the following definitions:

(1) The set of tasks which depend on library packages are called library tasks.

(2) The set of tasks which depend on the main program or on a task t or on a block or subprogram currently executed by the main program or the task t are called the dependent task of the main program or the task t.

The debugging system provides means for accessing these tasks, their current locations, the current location of the main program and the corresponding calling locations and calling tasks.

By this way, the programmer can get a complete overview of the current program state. Since the program structure can be very complex we introduce for convenience the notion of the actual location, i.e. the location actually inspected by the programmer. The debugging system provides operations for inspecting the actual location and for moving to another location thus making this location to the actual one. The static context of the actual location can be inspected

directly. This applies to the inspection of objects discussed in the next Section as well.

For implementing these operations the debugging system needs detailed knowledge of the runtime organization, especially of the tasking implementation. Therefore the implementation of these operations does strongly depend on the target machine. However, since we consider always the same compiler, these machine dependent parts can be hidden behind machine independent interfaces. By this way, the debugging system remains portable.

3.3 Accessing Objects

The possibility to inspect values of objects means great support for analyzing error situations. When, for instance, the program was abandoned with CONSTRAINT_ERROR because of a subrange violation the question concerning the relationship between a value and the subrange bounds arises. This example shows that accessing an object must also include the possibility to inquire certain attributes of the object or of its type, e.g. its constraints, if any.

The debugging system must solve three totally different problems when it allows the programmer to access the value of objects:

(1) The programmer will, of course, enter the name of an object in terms of the source language. The debugging system must then identify the object denoted by the name and find out where this object is currently stored. We refer to this place as the address of the object.

(2) The value of an object is represented by some bit string stored at its address. The debugging system must be able to interpret this bit string as a value of the type of the object.

(3) If the object is constrained, the debugging system must be able to access the values of these constraints. For instance, for an array, the lower and upper bounds of all dimensions must be accessible. This also applies to other attributes of the object or its type.

These three items are discussed in detail in the subsequent paragraphs.

3.3.1 Locating Objects

In order to ease the navigation through the current program state we introduced in Section 3.2 the notion of the actual location. We define here in terms of the source language which objects are accessible to the programmer in the actual location. So, by moving through the program (as described in Section 3.2) all objects existing in the program can be made accessible.

An object or package which is visible at a location or which could be made visible there by qualification is accessible at this location. Additionally, all library packages and all objects or packages which are declared in packages accessible at this location and their bodies are accessible at this location as well.

By this way the programmer may access (beside the objects visible at the actual location) objects which are hidden by an inner declaration and objects declared within the bodies of visible packages.

It seems necessary to allow the programmer to inspect the implementation details of packages during debugging although this violates the information hiding principle of the language. Perhaps it would be more appropriate to establish a protection mechanism which allows only the implementor or maintainer of a package itself to inspect its body. This could be achieved e.g. by the debugging system asking the programmer to enter the appropriate password when he wants to access hidden information or by hiding the information from the program library in general.

The locating of objects works in several steps:

(1) The name entered by the programmer is analyzed whether it denotes an accessible object. Ambiguities in naming caused by the source language, e.g. by overloading or hiding, must be resolved during this step.

(2) The definition of the object in the Diana representation of the program is searched.

(3) The program library is accessed in order to obtain addressing information for this object. This results in a pair consisting of the name of a frame and the offset of the object within this frame.

(4) To compute the address of the frame further information must be retrieved:

- If the frame is allocated statically, its address can be obtained from the linker listing.
- If the frame corresponds to the invocation frame of a subprogram or a block, its address can be obtained from memory of the interrupted program: the display vector or the static links must be inspected.
- Otherwise, the frame address is stored in a pointer object. The address of this object is computed in the same manner.

(5) Finally, the address of the frame and the offset of the object are added. So, the address of the object is obtained.

3.3.2 Representation of Objects

After an object has been located the debugging system must be able to interpret the bit string stored at its address as a value of the type of the object. For this purpose the debugging system must know the representation the compiler has chosen for this object.

Since the representation of objects is fixed by the type mapping module of each compiler individually fitting to its own runtime organization, we have here again the situation that the debugging system depends strongly on the compiler, but not so strongly on the particular target machine.

Some characteristic data about the representation of objects are stored in Diana, but this information does not completely describe the representation. For instance, for integer objects the debugging system must additionally know whether the numbers are stored as binary numbers or as binary coded decimals, whether a sign bit is present, whether negative numbers are stored as one's or two's complement, etc.

This shows again that the debugging system must access the program library to get complete information. Additionally some target machine dependent issues must be considered which may be hidden behind a machine independent interface.

3.3.3 Constraint Information

Constraints may but need not be static in Ada. Hence, for some constraints additional objects must be introduced which hold their values. The information which object holds which constraint is stored in the program library and must be inspected by the debugging system. Once the objects holding the constraints are known, Paragraphs 3.3.1 and 3.3.2 apply.

For the values of other attributes of the object or its type similar remarks apply.

4.0 Debugger Interactions

In this Chapter we discuss the implications of the interaction capabilities for the debugger-compiler interface.

The debugger is conceptually a (monitor) task with its own stack. If it runs as a post mortem dump analyzer no interaction between debugger and program under test is necessary except for the generation of a dump (file) on the programs side and the identification of that dump on the debuggers side. If the debugger is on line with the program under test different possibilities for its implementation are seen.

(1) The debugger is loaded together with the program under test getting its own memory in the address space of the program under test.

(2) The debugger runs in a different address space (parallel task with message passing).

The latter possibility has been chosen because it allows to handle host-target debugging in the same manner as host-host debugging. In the former case the debugger runs as a task on the host and the program under test on the target. In the latter case both run as different operating system tasks on the host computer.

For the location of breakpoints corresponding to source positions it is necessary to have the compiler build tables which contain the mapping of source positions to code addresses. The code to source mapping proposed in Section 3.1 is not sufficient because it does not allow to retrieve from a source position the code address which is executed first. The tables containing the source to code mapping are constructed in a similar manner to those containing the code to source mapping.

In order to set breakpoints on exception or tasking events the debugging system must have detailed knowledge of the runtime organization and tasking implementation.

For the modification of data the same information as for accessing objects is required. Additionally the debugging system must be able to interpret a string given by the user as a value of some type and transform it into a bit string to be stored at some objects address.

5.0 Conclusion and Prospects

In this paper we described the design and partially the implementation of a retargetable debugging system for the Karlsruhe Ada compiler. The most essential implication is that there is no need to have two versions of the program: one for debugging and one for real usage. Otherwise it would be very difficult, perhaps impossible, to guarantee that the instrumented version will produce the same error situations. On the other hand, debugging code must not be incorporated into the program to avoid overhead for the normal execution.

Hence, the debugging system has to work for programs which are not specially instrumented for debugging. All knowledge about the execution of a program must be retrieved from its memory when interrupted. Additionally, the debugging system may access information collected by the compiler, the linker and information about the runtime organization of the programs.

As has been discussed in Chapter 3 the target dependent features of the debugger can be concentrated in a few machine dependent packages making it easy to retarget the debugging system. The rehostability is given because the debugging system is written in Ada, as is the compiler.

We have shown that on this basis a reasonably working debugging system can be built: it provides all operations needed by the programmer in order to analyze an error situation.

A further advantage of this approach lies in its applicability to cross developed programs: The program is developed and tested on a host computer, but is used on some other machine, the target computer. If an error situation occurs during real usage, the program state (i.e. the contents of memory) can be written on tape and transported back to the host machine. There the environment is present for applying the debugging system (in the version for this particular target machine). By this way, error situations can be analyzed even for real time applications where errors usually cannot be reproduced.

On the other hand this approach allows to "field test" the program on the target with the debugger located on the host if an online communication channel exists between host and target.

5.1 Outlook

A shortcoming of most available debuggers (including the one presented) is that the program under test has to be stopped in order to analyze it. For some real-time applications this may be a serious problem. Therefor we are planning to develop debuggers which analyze the program behaviour in source level terms without disturbing or interrupting the running program on a target system. This is a most challenging effort with respect to real-time application programs.

The Relationship of Software Engineering and Artificial Intelligence

Gerhard Goos
GMD Institut für Systemtechnik

Abstract: In this paper we discuss existing and potential applications to the field of software engeneering of methods and tools developed in the area of artificial intelligence. We also indicate problem areas in the field of artificial intelligence which might be resolved by software engineers. The topics include programming by searching as a basic programming paradigm, the use of rule based systems and AI-languages, applications to rapid prototyping and program transformations. Furthermore the potential use of expert systems in software engineering is investigated.

1. Introduction

Software engineering as a branch of computer science is concerned with the theory and practical methods for efficient production of reliable and correct software in time. These issues comprise on the one side managerial questions of how to organize the software production process. On the other side software engineering is concerned with methods and tools for supporting the software life cycle, starting from requirements analysis up to the final acceptance test and maintenance phase. It is well known that presently the costs for modifications and improvements of software during the maintenance phase are much too high and amount to more than 50% of the total costs. The main issues in current research in software engineering are therefore on the one side questions of how to improve the productivity of programmers and the overall quality of the resulting product; on the other side we have the question of how to reduce the maintenance costs. It has been recognized that many of the problems stem from the fact that the design process very often starts from specifications which do not adequately reflect the intentions of the customer. Hence a specification method is required which helps to improve the results of requirements analysis. It would be even more useful when such a specification could be made executable so that the specification can be debugged by means of rapid prototyping before the design process starts.

Starting from this question interest has been created in methods of artificial intelligence amongst software engineers. Indeed, it turns out that AI-methods might help in this situation but might also have other applications in software engineering. In this paper we discuss such applications. At the same time it turns out that some methods of software engineering might be applicable in the area of artificial intelligence.

My co-workers Reinhard Budde, Peter Kursawe, Karl-Heinz Sylla, Franz Weber and Heinz Züllighoven have contributed to the ideas expressed in this paper.

2. Programming by searching as a program paradigm

One of the key issues in the design phase on all levels is the adequate break down of the problem in hand into sub-problems and the corresponding construction of the solution from solutions of the sub-problems. Any method for solving this question leads to a model of software construction called a *programming paradigm*. Many such methodologies have been invented and successfully applied like top-down design (abstract machine modelling),stepwise refinement, the use of models from automata theory, etc.

One methodology, *programming by searching*, has been rarely used in software engineering up to now although it has been proven very successful in artificial intelligence. Although this methodology does not necessarily lead to efficient programs it has a number of advantages:

- The programs developed according to this methodolgy comprise a part which might also be read as specifications of the problem.
- These specifications might be expressed in a way which is understandable also to the non-specialist.
- Initially the specifications might even be incomplete or contradictory. Such problems might be be interactively resolved by user intervention or by dynamically modifying the specifications.

The basic idea of programming by searching starts from the assumption that all possible solutions are known a priori (a potentially infinite set). These possible solutions form a *state space*. The solution(s) corresponding to the actual input data is found by successively generating the elements of the state space and by testing the generated state wether it is an appropriate solution. This *generate and test* methodology is also known as the *British Museum method*. It obviously works in practice only if the state space is sufficiently small. The method, however, shows already the basic ingredients of programming by searching: An algorithm for implementing this method consists of the following:

- an initial state,
- a set of rules for generating new states from a given one,
- a control algorithm for determining the rule to be applied next,
- a target condition characterizing the desired solution.

The method may be generalized by hierachically structuring the state space: Possible solution are no longer generated in one step but chains of states are generated which might be intuitively thought of as constituting successive approximations to the solution. The algorithm described above basically remains unchanged. Some specializations of this method, e.g the greedy method, are well known to the software engineer. They are characterized by the fact that the control algorithm and the rules generating new states are integrated in a specific way.

The method, also called *unidirectional search*, however, is of limited scope only. It requires that in each state the algorithm is able to determine which rule will definitely lead to the solution (if it exists at all). To remove this restriction we may consider a generalization of the notion of state: If each new state also comprises the information about all states generated so far we may restart the search at any former state if the last generated state appears to be a dead end (or "less promising"). We could even generate new generalized states by simultaneously applying several rules. An abstract implementation of this generalization uses a *search tree* instead of a chain of states as indicated in fig. 1. The generalized state consists of that part of the search tree generated so far. The control algorithm must now pick a node in the search tree together with a rule which is applicable to that node.

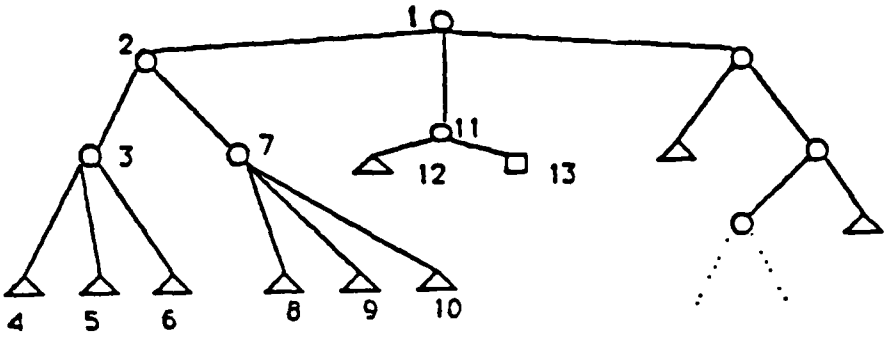


Fig. 1 A Search Tree

Figure 2 shows the complete search tree for the 4 queens problem. Except for the two solutions of this problem all other leafs are dead ends. Starting from the initially empty board (on level 0) there is one rule in this game indexed by integers $i,k, 1 \leq i,k \leq 4$:

From a node on level $i-1$ generate a node on level i by placing a queen in row ix, column k subject to the condition that this position is admissible (cannot be reached by any previously placed queen.)

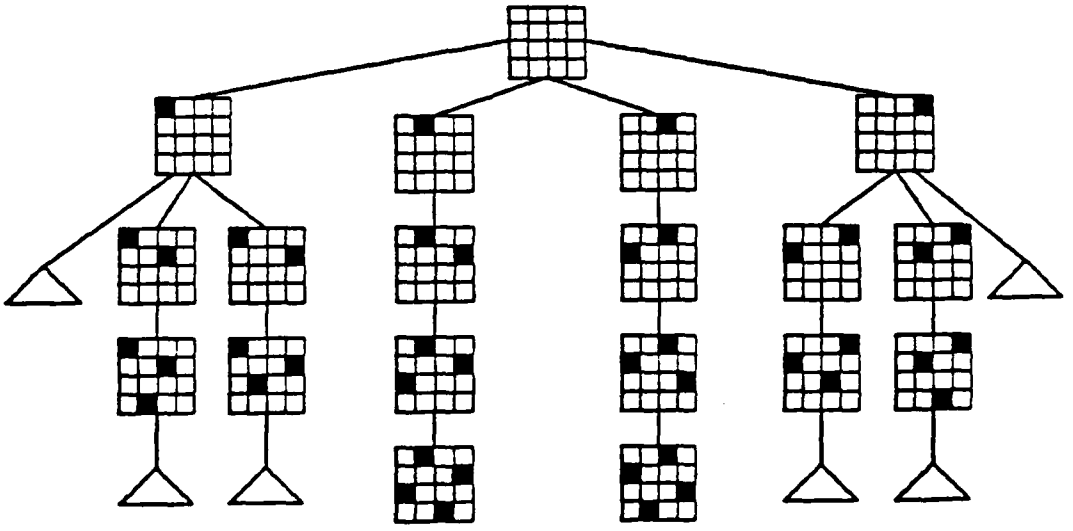


Fig. 3 The Search Tree for the 4 Queens Problem.

The notion of search tree leads to the fact that the final generalized state also includes the path from the initial state which leads to the solution. This information is often equally useful as the solution itself: In many problems the "solution" is known a priori but the way how to achieve it is the desired information. For example, in a diagnosis system the fault symptom is initially known but the possible sources of the fault are indicated by chains of malfunctionings leading to this fault symptom.

In a practical implementation of the control algorithm and the building of the search tree we are faced with a number of fundamental engineering problems:

- The indeterminism: which node to consider next
- The indeterminism: which rule to apply next
- The potentially large number of rules to be tested for applicability
- The combinatorial growth of the search tree and the associated storage problems
- The fact that the search tree may contain potentially infinite subtrees which do not lead to a solution. (Hence the search might never terminate although a solution exists.)
- The question how to identify dead ends as early as possible.

A number of control algorithms have been developed for dealing with indeterminism: Depth-first search, breadth-first search, the use of heuristics for determining the next node and rule, etc. The book [Nilsson 1982] contains an extensive treatment of such search methods. Most of those methods can be combined with recognizing cycles which are (partially) responsible for infinite search trees. Heuristics can be applied in two different ways: Either we introduce priorities indicating the order in which branches should be added to the tree; this idea may improve the speed for finding solutions to the extent that problems become practically tractable which are beyond the accessible computing power otherwise. Or we may definitely decide that a certain branch does not lead to a solution based on heuristic arguments. Whereas the former way does never exclude solutions - at least in theory -, the latter way may restrict the number of solutions being found since the heuristics might be wrong. Hence altogether programming by searching may lead to the following results:

- Success: A solution (or several/all solutions) is found together with the sequence of states leading to the solution
- Definite failure: A solution does provably not exist
- Failure: A solution has not been found because either the algorithm did not terminate in a given number of steps or because an unsuitable heuristics was used or because no further applications of rules are possible for other reasons.

Despite the problems mentioned programming by searching has a number of advantages compared to other programming methods:

- The strict separation of rules and control algorithm allows for considering the rules as an *algorithmic specification* of the solution process. This specification may be expressed in readable terms but is hiding all of the implementation decisions connected with the representation of the search tree and the control algorithm.
- The user may get interactively involved either for supporting the control algorithm in its decision making or for introducing state changes manually in case the system does not find an applicable rule.

Unfortunately the other side of the coin is that the algorithmic specification may be used in a restricted manner only due to shortcommings of the control algorithm.

On the other hand the interactive involvement of the user might be explored for dealing with incomplete or contradictory specifications or for extending the specifications on the fly, a direction which could never been followed in ordinary program design.

3. Rule Based Systems

Formally speaking the rules forming our specifications, often also called *productions* form a *derivation system*, $D=<Q,R,I,S>$ where

- Q is a decidable set of states.
- $\rho: Q \to 2^Q$ with $\rho(q)$ for all $q \in Q$ being a finite (potentially empty) subset of Q. We map Q into its powerset because $q \in Q$ represents a generalized state; there may be several components to which rule ρ may be applied with distinct results. We write $q \to q'$ iff $q' \in \rho(q)$.
- I and S are decidable subsets of Q representing the *initial* and *solution* states.

Derivation systems can be analyzed for certain properties. The most important properties in practice are the following ones:

- A derivation system A is *noetherian* if each derivation
$$q_0 \to q_1 \to \cdots \to q_n \to \cdots$$
starting at an intial state $q_0 \in I$ terminates after a finite number of rule applications.
- A derivation system D is *confluent* iff for each initial state q_0 which has a solution and for arbitrary successor states u,v of q_0 there are derivations $u \to \cdots \to z$ and $v \to \cdots \to z$, z a solution ($\in L$).

A noetherian derivation system has the interesting property that any control algorithm no matter how it proceeds will terminate after a finite number of steps because all search trees consists of a finite number of nodes only. A confluent derivation system has the property that the solution is unique if it exists at all and there are no dead ends: from each reachable state we may also reach the solution. Both properties are much desired but do not occur in practice as often one might wish.

A control algorithm for using a derivation system $D=<Q,R,I,S>$ is an algorithm as follows:

Initial State: $q := q_0 \in I$
  loop *select* a rule q with $\rho(q) \neq \phi$; exit if no such rule exists;
    select $q' \in \rho(q)$;
    $q := q'$; exit if $q \in S$
  end loop
Output: q if $q \in S$, no success otherwise

As discussed before it might happen that the algorithm does not terminate. The algorithm is indeterministic due to the two *select* operations. The combination of a derivation system and a control algorithm is called a ***rule based system***.

Rule based systems appear in many forms in AI. Historically they have been termed *production systems*. *Frame systems* and other variations subdivide the set of rules and partially allow dynamic changes of the rule set depending on the current state.

4. AI Languages

From its beginning the people in Artificial Intelligence have used languages like LISP instead of the more common imperative languages FORTRAN, ALGOL, PASCAL, ... It is yet difficult to analyze why these differences in language approach have occurred, notably since, by various extensions, LISP has taken over most of the features of standard languages although with different syntax. Significant features of most of the languages in AI compared to imperative languages are the following:

- Basically there are no program variables in the usual sense (although they have been mostly added through the backdoor). Computations produce new values by combining existing ones rather than by modifying them.
- Hence values like numbers from which new values cannot easily derived by combination play a subordinate role. The basic data structures are lists of dynamic lenght with dynamic typing of the elements. Internally these lists correspond to binary trees. Very often lists are used for representing what is known in logic as a *term algebra*. This use occurs in its purest form in PROLOG.
- All AI languages are based on an interpretive model which allows for dynamic reinterpretation of data structures as pieces of program. Although this is a very dangerous feature, it is very helpful, e.g., in manipulating the set of rules in a rule based system.

Current programming languages can be classified as

- imperative: Operations are expressed by statements which manipulate state variables.
- applicative: Operations are applied to expressions forming a larger expression. The notion of a variable does basically not occur. Some form of the theory of recursive functions is the underlying theoretic model.
- functional: A program is a set of functional equations. These equations express relationships between values and unknowns (mathematical variables); the execution of the program has to resolve equations. Most mathematical theories can be most easily transformed into this model; but the knowledge by the program interpreter for resolving the equations is sometimes specialized.
- logic: A program consists of a number of formulas in predicate logic together with a model how new formulas may be derived from the given ones.

In practice none of these language types occurs in pure form; e.g., ALGOL 68 was a mixture of an applicative and an imperative language. Certain developments in the area of logic programming languages starting from PROLOG are especially interesting because they allow for subsuming many aspects of the applicative and functional programming style.

Of course, since all of the non-imperative languages rely on an interpreter which is currently written in software it is very difficult to achieve highly efficient programs and to judge the efficieny from the program text without knowledge of the interpreter. Hence all these language styles are interesting for the software engineer mainly for two reasons: First, some of these languages are suitable for writing specifications and programs written in such a language can be run as prototypes before the actual efficient implementation is developed. Second, programs in such languages are sometimes very concise and easy to read and write, hence it is possible to develop programs in much shorter time; problems may become solvable which otherwise could not have been attacked due to the shortage of programmers' time. We demonstrate some of these considerations using PROLOG as an example language.

PROLOG [Clocksin 1981] is a logic programming language based on the calculus of Horn-clauses. A Horn-clause is an *implication*:

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow p_0$$

written in PROLOG as

$$p_0 :- p_1, p_2, \ldots, p_n .$$

Here we have $n \geq 0$; if $n=0$ we call the clause a *fact*. The predicates (or literals) p_i may have terms as paramaters as shown in the following examples. Logic variables, denoted by identifiers starting with upper case letters, may only occur as terms. Hence PROLOG essentially remains in the realm of first order predicate calculus. Certain built in predicates (*assert*, *call*, *retract*) allow for reinterpreting a term as as Horn-clause and cause second order effects useful for manipulating programs.

Simple examples of PROLOG programs are

```
human(sokrates).
mortal(X) :- human(X).
```

This program allows for estabilisting the result

```
mortal(sokrates)
```

in the obvious way whereas all questions *mortal*(*xyz*)? would be answered with *no* as long as it or *human*(*xyz*) is not established as a fact.

The use for prototyping and the short way of expressing problems may be seen from the following programs for algebraic differentitation. (*dif* (*E*,*X*,*DE*) means: *DE* is the dervative of expression *E* with respect to *X*.)

```
dif(U+V,X,DU+DV) :- dif(U,X,DU),dif(V,X,DV).
dif(U*V,X,(DU*V)+(U*DV)) :- dif(U,X,DU),dif(V,X,DV).
dif(X,X,1).
dif(Y,X,0) :- Y≠X.
```

The query

```
dif(a*x+b,x,L)
```

will be answered with

```
L=((0*x)+(a*1))+0
```

which is the desired answer but yet not simplified.

It is obvious that this program for computing derivatives is much shorter than anything which we could write in ordinary programming languages. At the same time it is much more readable. It therefore can serve as a specification which at the same time is executable, i.e. as a prototype for a real implementation.

Horn-clauses can also be considered as the rules of a rule based system. Viewed in this way we may ask what is the control algorithm underlying the execution of these rules. In PROLOG this control algorithm is depth-first search with backtracking: For estabilishing the validity of a predicate the definitions (Horn-clauses) of this predicate are searched in top-down; for each definition the predicates on the RHS are considered in order from left to right and their validity is established.

The use of depth-first search is unsatisfactory in both ways (but so would be any other control algorithm): On the one side it introduces a difference between the abstract understanding of Horn-clauses as specifications and their interpretation during execution; we have to distinguish between the *declarative* and the *algorithmic* interpretation. On the other side depth-first search is just one of the possible search strategies. If we want to use another one we have to simulate it on top of depth-first search. Fortunately also in these cases the rules may be written in a way which is easy to understand and hence the pro-

perties of Horn-clauses as specifications are mostly retained.

5. Software Engineering Problems

As mentioned earlier programming by searching poses some technical problems, e.g., dealing with a large number of rules and with a large set of state variables, which are typical engineering problems. In many cases normal methods of software engineering may be applied to solve these and similar problems.

A standard method is, e.g., to factor the state space and the set of rules in a hierarchical fashion. The result is a rule based system containing rules which themselves are rule based system. Also changing the abstract representation is very often helpful in reducing the size of states.

Another class of problems arises from the fact that in practice certain properties might be expressed as parts of rules and as part of the control algorithm as well. For example, in many cases it is posible to predict subsequent rule applications (at least with a certain probability) once a certain rule has been used. Sometimes it is possible to use the rules "backwards";i.e., if the solution is known but the sequence of steps leading to the solution is searched for we may start the search at the solution and work backwards to the initial state. Of course, this backward analysis, well known from other areas of mathematics and informatics, requires an adaption of the rules and it is not at all clear under which circumstances it bears advantages. Also combinations of forward and backward analysis have been successfully used in practice, especially in the form of "middle oout reasoning" where it is assumed that we know some intermediate state of the solution path in advance.

Yet another form of interaction between control algorithm and rules occurs if the applicability of a rule is known only with a certain probability. This situation occurs very often in expert systems for purposes of diagnosis. There are several strategies for distributing the handling of probabilities between rules and control algorithem.

The foregoing problem may also be considered as a special case of dealing with uncertainty. Uncertainty may occur on the rule side as discussed or on the date side. The required input data can only be observed up to a certain degree of reliability or very our time. In this case we may again use probebilistic rules or we may apply concepts of fuzzy set theory, etc.

All these questions are basically engineering problems in applying rule based systems. There exist a lot of proposals how to attack these problems - the interested reader is referred to the book [Hayes-Roth 1983] - which have been used in practice. But there does not exist a well developed and theoretically well founded basis for dealing with all these questions. Hence the practiceoner will find methods by looking into existing solutions; but at the same time the scientifically interested software engineer sees a waste area of research topics.

6. Applications to Software Engineering

Rule based systems may be used in many areas of software engineering basically as parts of a *software production environment*. In most cases these applications will take the form of an *expert system*, i.e., a rule based system with additional components facilitating the acquisition of rules (the *knowledge engineering component*) and the explanation of what is going on. Basically the value of such expert systems may have three different sources which mostly appear in some combination:

- During requirements analysis (but also during later phases of software development) it might be very helpful to view system analysis as the process of acquiring rules describing the intended computational model yet without the necessary control algorithm. This idea implicitly leads to a specification language, namely any language suitable for expressing rules. Psychologically the customer is much more inclined to support the system analyst when he declares himself as a knowledge engineer, and when he does not pretend to acquire all the fuzzy details which might change in a computational model anyway, but asks for the rules governig the present situation (it goes without saying that this method is only a trick which might nevertheless uncover all the necessary details).
- Expert systems may be used for performing tasks which are more easily expressed by rule based systems than by ordinary algorithms. Typical examples for this approach are applications to prototyping or for performing program transformations on all levels, including the transformation of executable or non-executable specifications into more efficient program descriptions. For example most of the existing catalogues of standard program transformations can be quite easily put in the form of derivation systems.
- Lastly it is often easier to resolve interactive tasks within the frame work of an expert system. Typical applications might include the tasks of configuration control based on large program libraries with modules in several versions and variants, support of tessting from the planning stage up to actual testing phase, or tasks occuring as part of project management. In many of these applications the extensibility of the set of rules may be used for starting with a relatively "unintelligent" system which then is gradually improved by adding new rules.

As an example for such extension techniques we might reconsider the differentiation program in section 4: This program can be immediately extended for dealing with arbitrary expressions by adding the interactive rule.

$$dif(E,X,L) :- write('please\ input\ the\ derivative\ of'),write(E),\\ write('with\ respect\ to'),write(X),read(L).$$

at the very end. When it turns out that certain types of expressions occur more frequently we may then add the appropriate rules for automatically handling these cases.

It is obvious that these techniques applied either singly or contined bear a large potential of fruitful applications to software engineering tasks. These possibilities have been yet only superficially explored, e.g. by developing program transformation systems. But many more immediately useful applications remain to be written.

7. Literatur

[Clocksin 1981]
W.F.Clocksin, C.S.Mellish *Programming in Prolog*. Springer 1981.

[Hayes-Roth 1983]
F.Hayes-Roth, D.A.Waterman, D.B.Lenat *Building Expert Systems*. Addison-Wesley 1983.

[Nilsson 1982]
N.J.Nilsson *Principles of Artificial Intelligence*. Springer 1982.

in: Proc. of the IFIP TC2 Working Conference on DATABASE SEMANTICS;
R. Meersman, T. B. Steel (Eds.), Hasselt, Belgium, Jan. 1985,
North Holland

OBJECTS AND ABSTRACT DATA TYPES IN INFORMATION SYSTEMS

Erich J. Neuhold

Institut fuer Angewandte Informatik und Systemanalyse
Technische Universitaet Wien
Paniglgasse 16, A-1040 Wien
Austria/Europe

Future generations of database systems will have to support a much wider variety of data objects than today's systems. Texts, voice, drawings, and pictures will have to be handled in an integrated fashion together with today's record oriented data in hierarchies, networks and relations.

To achieve this goal, a knowledge based, object and abstract data type oriented approach is proposed and consequences for the database management system, the information system design tools, and the system dictionary are discussed.

INTRODUCTION

Data base systems have had a tremendous impact in commercial data processing. The very valuable resource "data" of an enterprise has become accessible in an unified way to different company sectors, be it personnel, finance, manufacturing, marketing or sales. Centralized data bases and centralized processing of these data by the different enterprise components was a rational way of handling the necessary tasks.

With the availability of powerful miniprocessors this picture began to change. Computers started to appear in the different sectors of a company. Local processing and even interactive use of the computing resources became widespread. As a consequence local data storage increased and local data bases containing supposedly only data of local interest were established. However, data and the information represented by them are valuable assets of the whole enterprise and usually also needed elsewhere in the enterprise. As a consequence of distribution either redundancy of data storage with all ensuing problems of data consistency results or remote access to these data becomes necessary, i.e. multiple remote data base use in a single program becomes the rule, and problems of recovery and consistency will appear again. In addition dependencies on the location and the specific content of these data bases in a single program will practically inhibit to move data bases or individual parts of data bases from one location to the next. A solution to these problems is offered by distributed data base management systems, e.g. POREL [1], SIRIUS/DELTA [2], SDD-1 [3] or R* [4], that have been developed over the past few years. They handle consistency, location dependencies, reliability and recoverability in a user transparent fashion and provide in this way ease of access and resiliency toward communication or execution failures.

Recently, however, workstations and personal computers have become widespread and more and more people do their processing at least partially on their own office or home desk. Local data storage again is of importance for many of those applications as otherwise remote access or remote processing with all ensuing remote location problems has to be chosen for every task. Extending distributed data bases to include workstations and personal computers provides again a solution to the remote data manipulation problem and automatically solves the serious problem of limited storage resources in workstations and personal computers.

The widespread use of workstations, intelligent terminals and personal computers, however, has dramatically changed the kind of work a user expects his system to perform for him. Mail systems, multiple windows, graphics, even voice and video are expected to be offered in an easy to use manner. Icon oriented representations and flexible control facilities, e.g. mouse, pen, finger or even eye, are becoming available, soon to be enhanced by a multitude of knowledge-base oriented expert systems. Distributed and centralized data bases up to now are only able to handle effectively what has become to be known as formated data, i.e. data more or less directly derived from the record/field oriented interfaces offered in conventional file systems. The new kinds of data - which frequently are referred to as object oriented data - will require new concepts to be incorporated into data base management systems which will enable the users to access, manipulate and store those types of data effectively with at least the level of control embedded in today's DBMS's.

Interestingly enough two fields of computer science have dealt with this new type of data already for a considerable amount of time. In artificial intelligence the manipulation of 'objects' has been one of the basic features, but in practically all systems these objects are only manipulated in main memory. Sometimes primitive file systems are used to enhance the permanence and size of the data the system is capable of handling. Only recently with the coming of age of artificial intelligence the need has become paramount to manipulate large amounts of data that have to be shared and should be consistent and recoverable in the event of system failure. These requirements directly lead to data base systems where the AI community has found out that the type of objects handled there was insufficient to solve their
problems in practice. As a consequence of these requirements data base research and development have oriented themselves to 'object' data bases and considerable resources are spent to produce solutions in as short a time span as possible. Recently a well attended conference on Expert Data Bases [5] was organized and its proceedings contain numerous papers which illustrate some of the problems and solutions in this relatively new field.

The other area of computer science that dealt with 'objects' for a considerable amount of time tries to add 'meaning' to the data kept in data base systems in the sense, that the semantic of the data to be stored in the system is used to both: design the necessary data types (object types) and storage and access structures for a specific data base, and to develop and sometimes build right into the system operations for the manipulation of these semantically defined objects. [6, 7]. Originally the assumption in these systems was that the objects and operations would describe the universe of discourse of an enterprise in such a fashion that an actual data base could be derived either manually or semiautomatically. This data base later would be used in the conventional fashion either via interactive data

manipulation languages like SQL or navigational as for example in IMS or CODASYL. This idea, however, has to be changed if the users themselves are to be allowed to work with objects directly. It becomes necessary to build a data base management system that offers a semantic interface not only to the designer but also to all the users - humans and/or application programs - of the data base.

In this paper we shall now investigate the architecture of a future data base system that may be used to store and manipulate semantically meaningful objects, be they employee descriptions, product descriptions, letters, telephone messages, blue prints of buildings or machines, or videos for educational or promotional purposes. The most important consideration here has to be that all these data together represent the information source of an enterprise and therefore should be handled homogeneously, consistently and reliably by the system for its multitude of users.

THE SYSTEM ARCHITECTURE

In Figure 1 an information system is illustrated that has been designed to handle the requirements developed in the introductory chapter of this paper.

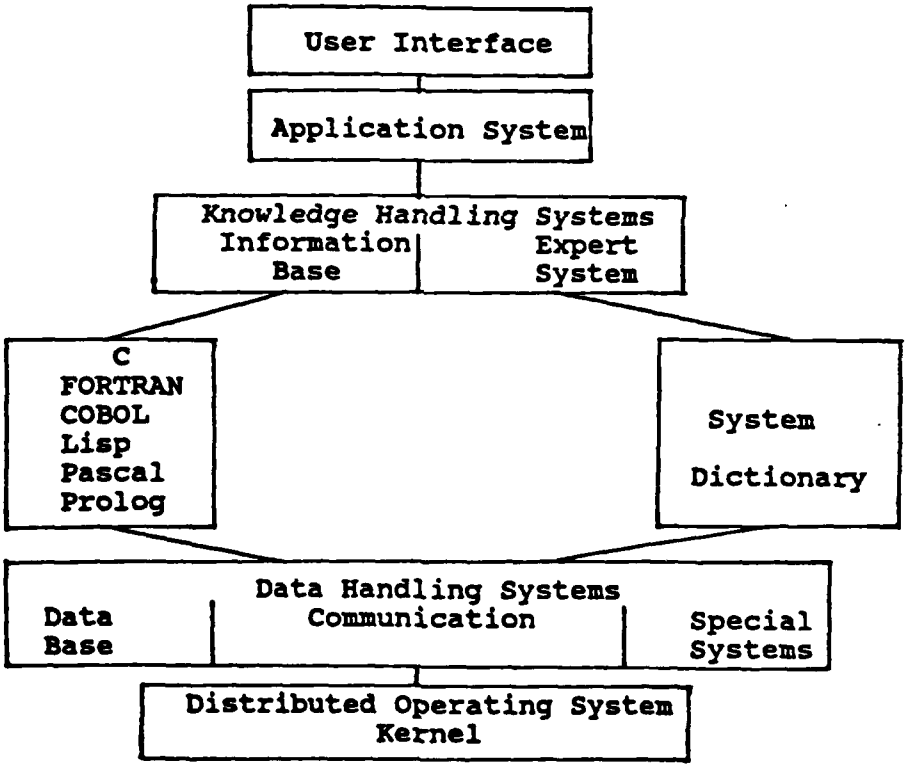


Figure 1
An Information System Architecture

The operating system kernel represents the lowest layer of the software architecture. It provides for data storage, process management and communication in the distributed (local or wide area) environment of our system. A more detailed explanation of the system can be found in [8] but a brief explanation here is in order.

The data storage must provide storage for a wide variety of data types but it must also provide sharing of data with flexible locking and recovery functions. Currently there still exists a separation between real/virtual main storage and file storage. The artificial intelligence community for example primarily relies on little shared mass storage as a depository for its complex data structures whereas the data base community concentrates on efficient access pates and storage concepts on disks but restricts itself to rather simple data object structures. In the future main storage and file storage will have to grow together not only on a single machine but also on local and wide area nets. Effective use of such structures will require knowledge on the type of object to be stored to be embedded directly into the storage system.

Data communications will play a central role in the network oriented architecture of all future systems. Different systems will have different requirements on the locality, speed, security, reliability and cost of communication. Local area, long-haul, broadcasting, point-to-point networks come to mind and frequently will have to work together in a single environment that undoubtably will contain computers of quite different size and architectural makeup, both with respect to hardware and to software.

Process management in the distributed environment of future systems will have to work intimately with both the data store and the communication facilities. Multiple processes will be involved in even relatively simple user tasks. They will have to cooperate across the network for retrieving, manipulating and storing the data of the system. Failure recovery from process, network, or storage break down will be essential for an acceptable reliability of our system and has to be built right into the operating system kernel in order to be available to all the other higher software layers in a coherent and consistent manner.

On top of the operating system kernel special data handling subsystems can be found. Data base and communication systems will be supplemented with special systems as for example special hardware for high level language systems such as Lisp, Prolog or Pascal. Other systems may be provided for CAD/CAM tools or for automatic control systems. In all these systems the emphasis will be set on handling data not on interpreting them as is done in the knowledge handling subsystems of the higher architectural layers.

Data base systems will have to manage distributed and shared data of quite complex structures. Different users - humans or programs - may have different views on these data, their access must be coordinated and controlled. Reliability and consistency of the data has to be embedded to such a degree that human controlled error recovery will practically never be necessary as the envisioned complexity of these systems would make such a task extremely complicated if not outright impossible.

Communication systems will provide a variety of services to the different system components. Telephone communication, mail, videotex, and video signals will be added to todays widely used communication types like file transfer/access, message delivery and sensing/control.

The subsystems of this layer will, however, still emphasise the handling of data not the handling of information/knowledge. Data will have known structures but it will not be known whether an individual

structure will be an employee description or the route map of a delivery truck. Generalized behaviouristic knowledge of course will be required as for example that same part of a structure (representing an employee) will be needed twice a month - for salary calculation - at a specific central location. This would allow a data base system to decide for example to maintain a backup copy of the structure at precisely that location instead of some other randomly chosen one.

The knowledge handling subsystems will actually understand the type of information even the individual information items that have to be manipulated. An information base system, for example, will - of course in an abstracted manner - know that a specific data structure represents an employee, what part of the structure is to be used when talking about the employee (its name, identifier, key?) and what other information about that employee is in the system. It will for example understand that an employee always has to be associated with a department, that he has to have some specific salary, or may be married. In addition such an information base will have to know about the kind of operations allowed by the using programs or humans against the individual date item. Only in this way will consistency, security and recoverability be ensured at an acceptabel level. For example a get-married operation for employees will provide the mechanisms to change the marriage status of exactly two employees, adjust their tax deductions and may even initiate reassignment procedures if both persons work for example in a direct-line management relationship and company policy does not allow for such situations.

To achieve its task an information base system has to be able to manipulate both structured data and programs working on these data. For this purpose it will access the data base on one side but use programming language subsystems on the other as the operations obviously will have to be implemented in one or the other of our programming languages. In Figure 1 we have also identified the system dictionary as a facility to provide the necessary meta-data which describe objects types and object structures, interrelationships between objects, and programs to manipulate specific object types or even individual objects. In conventional data base terminology the System Dictionary contains the schema description of the data base but this description of course has to be expanded considerably as in our system it also has to contain semantic information on data and operations in order to support the mechanisms of the information base.

The expert system component in Figure 1 actually is meant to describe all the mechanisms needed to support specific expert systems - e.g. a geological assistent or a network-configuration expert - which themselves are part of the application layer. Inferencing engines, interpreters but also strong links to the information base, the system dictionary, and programming environment will be required to provide the flexibility and large knowlege base to make the expert system truly successful in the demanding markets of the future.

The application system layer finally ties all the facilities offered by our distributed system architecture together. Using the information base, the expert system, programming language environments as well as the system dictionary the individual application will be able to concentrate on solving problems instead of dealing with the many complex issues of ensuring that data are handled in meaningful ways only, as these tasks - i.e. adding

semantics to the data - will already be achieved at the lower levels of our system.

The user interface will directly reflect the multi-media facilities of work stations and personal computers. Voice and graphic input and output will be available besides textual representations. A user will be able to interface with one or more applications concurrently in a personalized manner. For this purpose an expert system based on stored user characteristica will be available to the application packages. It will model learning, practice even forgetting behaviour of humans and select between a number of possible representation schemas on the basis of personal preferences.

To achieve this system goal the representation of the meaning of data and data operations is of central importance. Both the AI world and the data base world have developed concepts in this direction to ultimately allow the handling of information instead of the handling of data. Currently many investigations try to combine and unify these schemas to provide for the combined benefit of large information stores and expert system technology.

In the remainder of this paper we shall outline an approach based on abstract data types that was originally developed at the University of Stuttgart. Detailed descriptions can be found in [9, 10, 11, 12 and 13] but other proposals also exist as can be seen for example in the conference proceedings of [13] and [5].

OBJECT ORIENTED DATA BASE SPECIFICATION

When specifying an object oriented data base either for data base design or - if the system provides an object oriented interface - for user interface definiton, four important aspects of the data base system can be identified and have to be modelled:

1. Objects, type classifications and object structures
2. Interrelationship between objects
3. Operations, operation classes and operation structures
4. Dynamic interrelationships between operations.

In the following we shall use a small example to illustrate the use of these concepts for representing the information contained in an object oriented data base.

Object Classification Schema

Every material or immaterial entity of the real world that is to be represented in the data base is considered to be an object. This (large) set, however, has to be structured in order that objects with similar properties can easily be identified and manipulated in a homogeneous fashion. In Figure 2 an Object Classification Schema illustrates that the class OBJECT can be subdivided into object classes PERSON, COMPANY and THEATER CLUB. The PERSON class is further subdivided into EMPLOYEE, SUPPLIER and ACTOR. THEATER CLUBs may concentrate on modern or classic theater, COMPANYs may be trading companies or manufacturing companies etc. Notice that the subclasses of a class may be overlapping, e.g. an employee may also be a supplier and/or actor.

In the specification each of the object classes will have a verbal description attached , explaining its purpose and at least some of its most important properties.

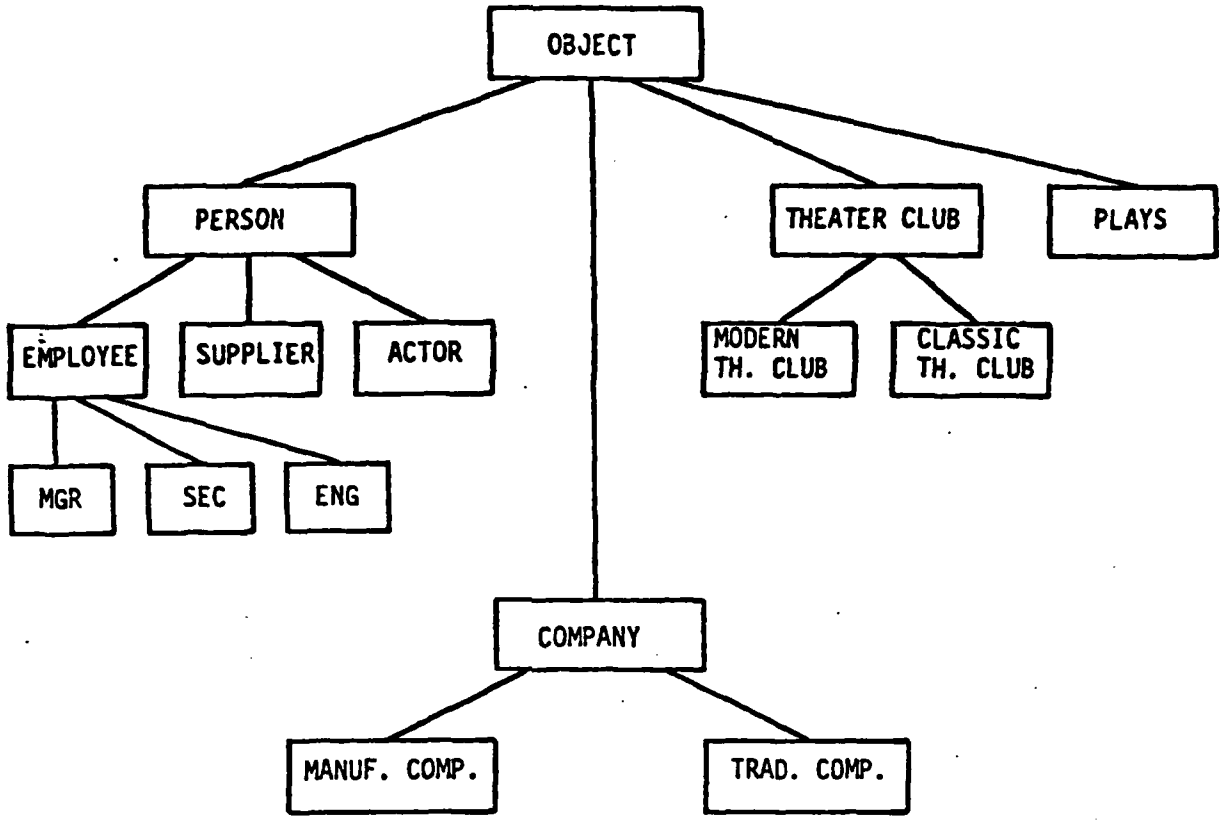


Figure 2
An Object Classification Schema (partial)

In any real system with hundreds or thousands of object classes it will be of utmost importance to present to the user and even the data base designer only a meanigful subset of these classes at any point of time. For example if a user currently is concerned with theater the object classes ACTOR, THEATER CLUB and PLAYS and their subclasses would be of a prior interest and therefore should be displayed, suppressing all the other schema components.

But how does the system decide which part of the schema it should display? Here the other descriptive information supplied with the data base has to be used. For example such information will identify that actors are members of a theater club, that plays are performed by theater clubs, etc. A knowledge based system using expert system technology will be able to make such a selection and present to the user at least initially an easy to understand subset of the total schema.

Object Structure and Relationship Schema

In Figure 3 an example of an Object Structure and Relationship Schema is partially given. Whereas an object classification schema only identifies classes and subclasses of objects we now are able to describe the structure of complex objects and their relationship to other objects.

A MANUFACTURING COMPANY is a complex object containing other objects like TEAM, DIVISION, MACHINE but also EMPLOYEE NAME or WORK UNIT that are related again to each other. A more detailed discussion of this

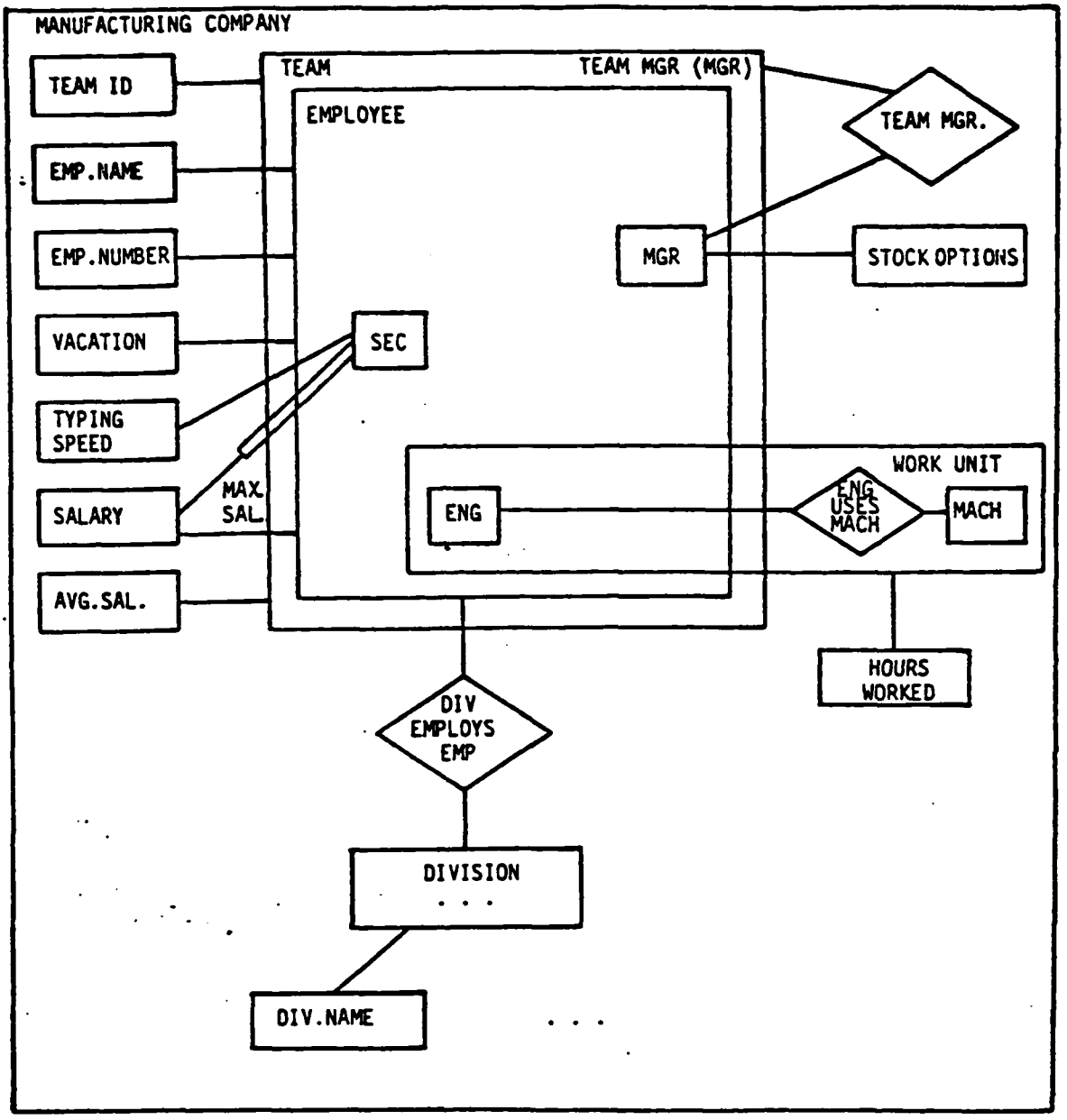


Figure 3
Object Structure and Relationship Schema for a MANUFACTURING COMPANY

schema type can be found in Furtado/Neuhold [12] but again any realistic application will lead to very large structures and a knowledge based system will have to select for presentation these parts that are of interest to a user/designer at some specific point of time. For example, if we want to talk only about employes in general without discussion of the properties of its subclasses, then the displayed schema could well be the one shown in Fig. 4. Notice that because of the subsetting/abstraction involved the description

becomes identical for manufacturing and trading companies and consequently is displayed for the super-type COMPANY directly. We have to remember here that the object classification graph also exists and informs the user or designer about facts like subclasses of COMPANY or EMPLOYEE so this information does not have to be displayed here.

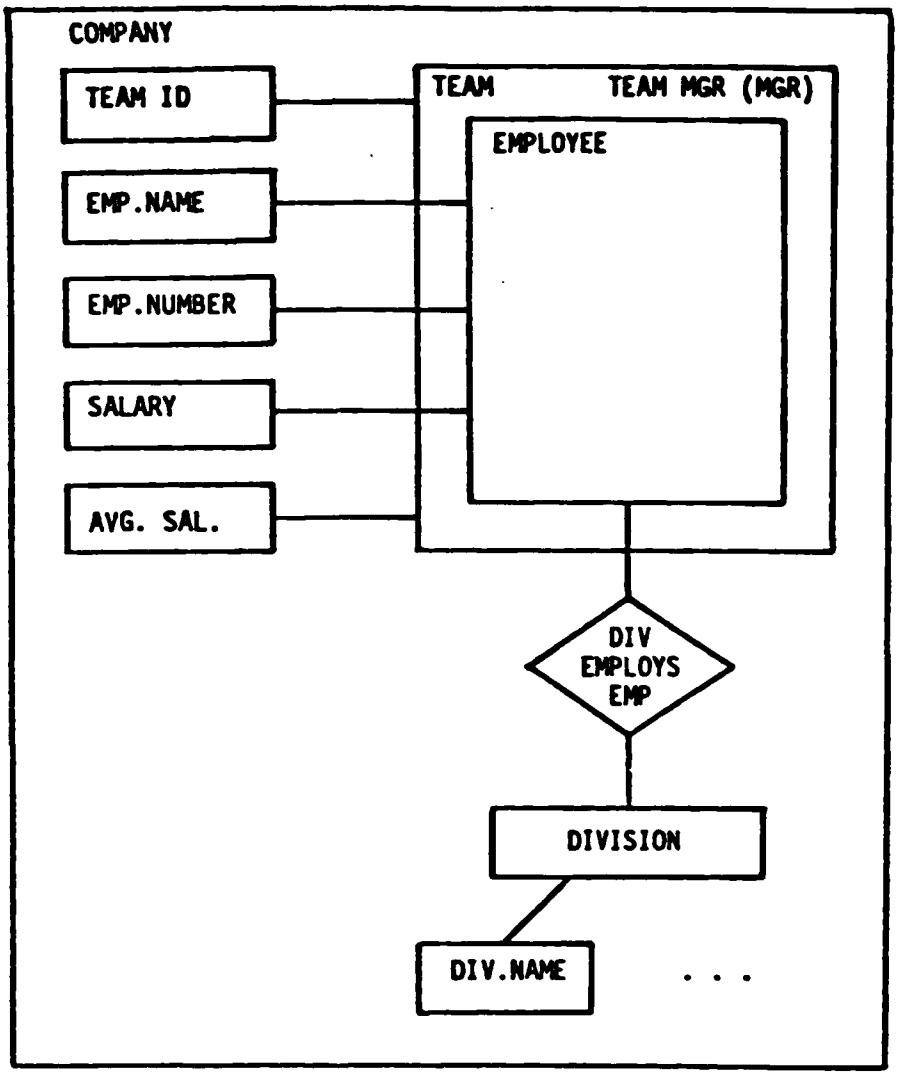


Figure 4
Object Structure and Relationship Subschema for EMPLOYEE IN COMPANY

Operation Classification and Structure Schema

So far we have only described the data oriented aspects of our system. However, in order to specifiy a complete information handling system we also have to identify the operations available, their interrelationship and their properties. For this purpose the Operation Classification and Structure Schema was developed. In Figure 4 we illustrate a (partial) schema that describes the properties and structure of an operation HIRE MANAGER that will work with the object and object classes identified in Figure 2 and 3.

An operation is given a name, e.g. HIRE MANAGER, and a parameter of some type, here e.g. PERSON, is specified. The operation description

then allows to identify preconditions and postconditions together with the operation body - the activity description. The operation may use other operations supplied with the system, here e.g. HIRE EMPLOYEE, ENTER MGR SAL etc.

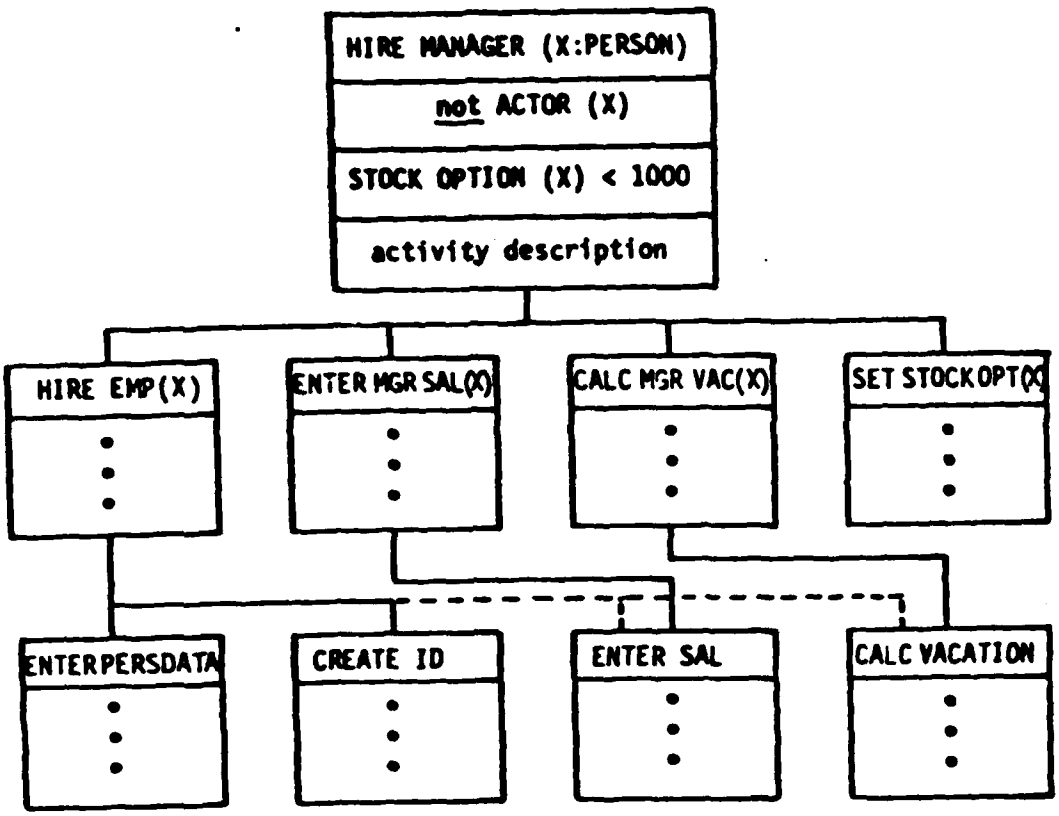


Figure 5
Operation Classification and Structure Schema

Using dependency structures as for example introduced in STUDER [14] we are also able to represent conditional execution, parallel execution, choice, etc. graphically such providing the user with a more explicit description of the interdependencies of operations. Note that the graph displayed is oriented toward the operation HIRE MANAGER. The operation HIRE EMPLOYEE which also will be used to hire secretaries and engineers for example will not use the suboperations ENTER SALARY and CALCULATE VACATION as these properties of a manager are handled by separate manager oriented operations ENTER MGR SAL, ENTER MGR VAC, which in turn utilize those suboperations. Operations can be very general. They are not restricted to handle only information kept in the data bases. For example the operation CREATE ID will not only select an employee number but also produce a badge in a truly integrated system it could even use other suboperations to take a picture of the new employee and produce his id-card.

Behaviour Schema

In our specifications we so far have described objects, object classes, object structures and object interrelationships as well as operations, operation structures and operation interrelationships. These pictures, however, represent only the static characteristics of the system. Of course new components may be added, old ones may be deleted but what is not represented is "the run-time" behaviour of our model.

In Figure 6 (some of) the activities surrounding the hiring of a manager are displayed. Of course for each of the operations an Operation Classification and Structure Specification and for each of the object classes on Object Structure and Relationship Specification would exist and could be displayed interactively to the user/designer. Notice that the operation PROCESS STOCK OPTION will be triggered by the HIRE MRG operation in the sense of a data flow diagram. It performs the necessary company actions to actually grant the option. If this has happened and the complete description of the person has been entered in the system then a congratulatory letter is sent to the new manager.

Like for the other diagrams a knowledge based system is required to present only those parts of the system behaviour that is of relevance for the current activity of the user/designer. For example the operation PROCESS STOCK OPTION will also be executed whenever a new stock option is granted to an employee but this is not shown in the displayed diagram.

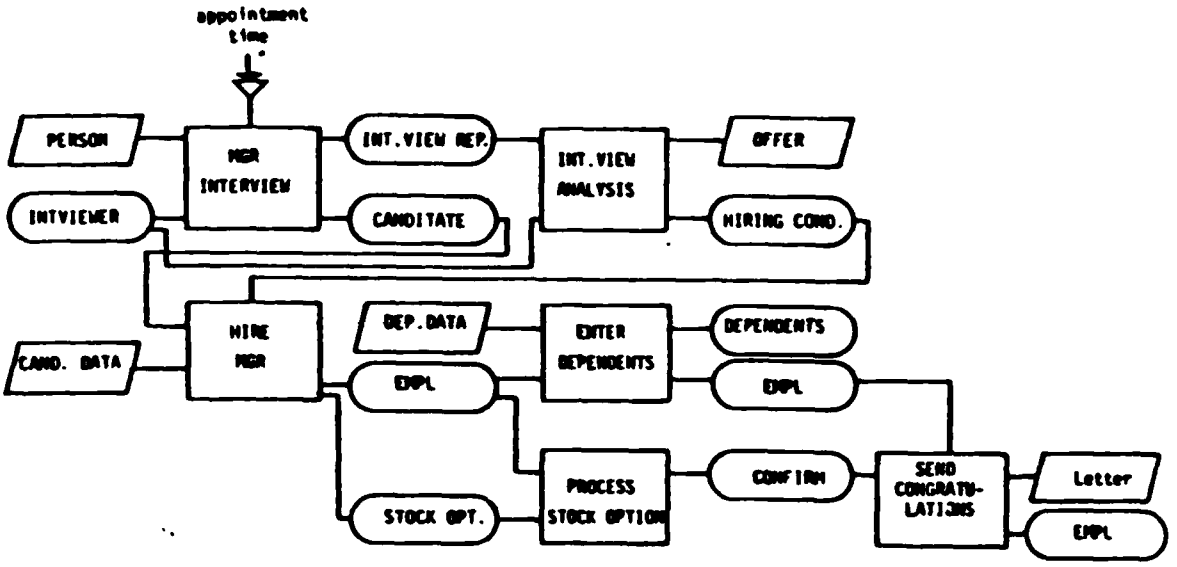


Figure 6
Behaviour Schema

SUMMARY

In this paper we have attempted to argue that future data base systems have to store and handle interpreted data - i.e. information - and that consequently current systems have to be extended with object and abstract data type oriented concepts. We have then proposed an architecture of such a system and discussed how the designer respectively user may be informed about the objects and operations represented in the data base. We have only given an overview of the respective features but more details can be found in the references [8] through [12]. It is important to realize, however, that the system which actually provides the information to the designer respectively user will have to be very flexible and will have to employ knowledge based and expert system techniques to present only the actually needed environment to the user and select for presentation formats adjusted to the preferences and knowledge of the individual user. These concepts, however, are currently still subject of research and it will probably take some time until they will become available in more than small experimental systems.

REFERENCES

[1] Neuhold E. J., Walter B.: POREL: A Distributed Data Base Management System, in: H. J. Schneider (ed.) Distributed Databases, North Holland, Amsterdam, 1982.

[2] Litwin W. (et al.): SIRIUS Systems for Distributed Data Management, in:
H. J. Schneider (ed.) Distributed Databases, North Holland, Amsterdam, 1982.

[3] Rothnie J. B. (et al.): Introduction to a System for Distributed Databases (SDD-I), ACM TODS 5, 1, 1980.

[4] Daniel S. D. (et al.): An Introduction to Distributed Query Compilation in R*, in: H. J. Schneider (ed.) Distributed Databases, North Holland, Amsterdam, 1982.

[5] Kerschberg L. (ed.): Expert Database Systems, Proc. First International Workshop on Expert Database Systems, Inst. of Information Management, University of South Carolina, 1984.

[6] Tsichritzis D. C., Lachovski F. H.: Data Models, Prentice-Hall, Englewood Cliffs, (1982).

[7] Olle T. W., Sol H. G., Verrijn-Stuart A. A. (eds.): Information Systems Design: A Computative Review, Proc. of IFIP TC8 CRIS I Conf., North Holland, Amsterdam (1982).

[8] Derrett N., Neuhold E. J.: Information Systems - The Next Ten Years, in: Proc. of JCIT 1984, IEEE Computer Society, Silver Spring, 1984.

[9] Schiel U.: A Semantic Data Model of Conceptual Schemas and their Mapping to Internal Relational Schemas (in German), Doctoral Thesis, Univ. of Stuttgart, 1984.

[10] Studer R., Horndasch A., Yasdi R.: An Approach to (Office) Information Systems Design based on Generalized Net Theory, in: TFAIS 1985, North Holland, Amsterdam, 1985.

[11] Studer R., Horndasch A.: Modelling Static and Dynamic Aspects of Information Systems. in this volume, 1985.

[12] Furtado A., Neuhold E. J.: Formal Techniques for Data Base Design, Springer Verlag, Berlin, 1985.

[13] Sernadas A., Bubenko J., Olive A. (eds.): Theoretical and Formal Aspects of Information Systems, Proc. of TFAIS 85, North Holland, Amsterdam, 1982.

[14] Studer, R.: Functional Specification of a Decision Support System, in: Proc. of VLDB 79, Rio de Janeiro, 1979.

MANAGEMENT OF SOFTWARE FOR LARGE TECHNICAL SYSTEMS

H. Halling

Abstract:

For the design, construction and operation of large technical systems, software plays an important role. In this presentation some aspects of managing software for such systems are discussed. The different classes of software during the project phases and within the different hierarchical levels of a control system are outlined and their relations to proper management are shown. In addition, the problems of purchasing software and estimating the required time, budget and manpower for a project are discussed. Emphasis is placed on practical aspects and examples are presentd.

Introduction:

During the last few years I have heard of many projects, where the top project management had developed an uncomfortable feeling about computer based activities and in particular about software. This was especially true for all projects where I was personally involved.

I think that this feeling has two main sources. Firstly, most of the project leaders never wrote software themselves, so that an understanding of the kind and amount of activities which lead to a useful product is missing. Secondly, it is a fact, that the control of the progress of software development, even on a high level with only a few milestones, is difficult. As a consequence, software and software producers cannot expect to find enough credibility and trust.

In my opinion, only the attempt to demonstrate that software products play an important role within a project and that software can be planned and kept under control in order to fulfill its tasks within the project, will lead to better understanding and trust. The following is an attempt to provide an overview concerning software and its management for large technical projects, keeping in mind that nowadays there is nearly no project activity where software is not involved.

Management of software includes planning, control and correction of all kinds of software activities through all project phases and levels of the control system. Activities are requirement analysis, evaluation of products, purchasing, organisation of training, hardware layout, installation and maintenance of packages, system tuning, preparation of user-friendly interfaces, structuring of databases, definition of internal standards and finally producing software products where necessary.

Clearly nobody can expect all these activities to be treated in detail in this presentation, but some of the interesting items are picked out, especially those which have changed recently, or which are in a phase of rapid development.

Project phases and their relation to software:

Fig. 1 shows the important project phases and the software related to them. Software which is relevant for all phases is project management software, controlling deadlines and budget in correlation with the achievements of single activities and organisational support of the project.

The figure also shows that the software which is actually produced during the project and the tools and utilities needed for these activities are only a part of all the packages involved. For other fields of engineering like mechanics, electrical engineering, thermodynamics etc. packages exist for the different project phases. The results of such activities may be a valuable input for the process of producing software for the control system.

Some of the packages like the control of cost flow or deadline control must be accessible by everybody responsible for a part of the project during all of the project phases. This implies a common database and communication system. At this point, the question arises as to whether it is preferable to connect the relevant team members 1. via dedicated terminals to a dedicated management support system, or 2. to link the individual and most probably different data processing systems into a network running distributed management software and database systems. Solution 1 is simple and relatively cheap, but may later result in a variety of independent dedicated communication stars. Solution 2 may cause an incredible amount of work, may degrade the performance of smaller sytems drastically and may end up with a different project where the programmers have to learn harsh lessons about standards and compatibility without achieving anything for the original project. Up to now such software is mainly running on mainframes, most of which support distributed terminal services. Another package used throughout the phases deals with organisational matters, including configuration control, documentation libraries, organisational communication etc.

Most of the software packages used in the earlier stages of a project are tools. From the very beginning it is extremely efficient to involve people familiar with the process to be controlled (like physicists, chemists, process engineers, technicians etc.). This implies the need to provide tools offering a proper man machine interface.

During the design phase, many large packages are available which can only be used by people familiar with the technical background concerning radio frequency, layout of a reactor core, temperature control, mechanics etc. Some of these packages apply finite element techniques, Monte Carlo simulations etc. and may run for hours on a 32 bit minicomputer system, because mainframes often do not guarantee short response times for interactive use. For these packages careful time scheduling and night shifts have to be organized and often terminals "near to the bed" are desirable.

Regarding fields like mechanics, electrical engineering, electronics etc. CAE/CAD/CAM techniques are being applied. These packages are expensive and in addition require special hardware and training. Harmonisation within a project is delicate (see solutions 1 and 2 above). At this point let me give an example how dedicated packages for a special engineering field might yield inputs to the programmer team working on the control system. Nowadays, if one is working on the design of an accelerator, a set of packages exists in order to evaluate design variants or the impact of misalignment etc. Such a package can be used by the control people to calculate the accuracy of set points and the impact of deviations (Fig. 2). Thus the transfer of information between the two parties took place by handing over a simulation package used by both to find out about different aspects. I am not going to discuss further details concerning the software production phases and their tools, because this is a topic this audience is familiar with.

Structure of control systems and its relation to software

Due to a variety of reasons, the programmable elements at the levels various of a large control system are quite different (Fig. 3). On the lowest level programmable logic controllers (PLCs) are number one for interlocks and sequencing. Due to the specific problems and the kind of personnel involved, problem oriented programming dominates. Positive results are predictable programming time and timing sequences which can be calculated. Furthermore the robustness and interactive testing as well as extended features concerning graphical programming and improved functionality are advantageous. The weak point in the past was the lack of proper communication between PLCs and higher leves. Meanwhile the situation is changing.

Other elements of the lowest level are microcomputer based data acquisition systems or systems controlling complex peripheral units. Nowadays such systems are often based on micros and personal computers and programming is usually similar to minicomputer programming concerning the types of languages and the language environment including operating systems and communications.

The connection of low level elements between each other or to group control units is done by serial field buses well suited to the harsh environment. As fas as software is concerned, there is no standard communication at this level up to now.

The so called backbone communication system is on the way to be standardized at least to layer 4 (transport layer). This requires up to 100 kbytes of interface software but is the way to standard communication between low level elements and minicomputer systems which are the basis for the elements of higher levels (Fig. 4).

Functionally the higher levels can be divided into three groups. Process control computer systems are responsible for higher level control and contain realtime software either stand-alone or with a proper operating system and realtime databases.

The second group is workstations or console systems mainly responsible for the high level man machine interface. This is the place for interactive graphics, menu libraries and in the future maybe even expert systems.

The third group is large minicomputer systems where the functional packages of the higher levels can be run. These include large application program packages, database management, all kinds of utilities and the communication software for intersystem communication and communication with higher level computers like mainframes or number crunchers. This minicomputer system runs the typical software for development, configuration control, longterm database management etc. (Fig. 5).

Evaluation of software products

The amount of manpower and the experience of the programmers required in order to produce software practically forces implementors to purchase software products. This is evident in case one needs operating systems, database systems, compilers, utilities or tools. But meanwhile even application software or at least parts of it can be purchased and tuned to meet the special requirements of a certain application. The only problem is that unlike hardware, a datasheet for software does not help too much. In general the products are so complex, that with present technology only a careful evaluation of the product leads to useful decisions.

Such an evaluation must be carefully prepared and result in evaluation specifications which contain what should be measured, how this should be done and in which order and how the results are to be documented etc. This ensures that different products are treated the same way. Fig. 6-10 show some examples concerning the evaluation of a relational database and a distributed operating system.

Tools are very difficult to evaluate. The ideal evaluation would be to run a small project applying the tools to be evaluated. The main advantage is that several programmers take part thereby reducing individual human preferences and also testing the cooperation between the group members. Definitly the introduction of Ada should follow this line.

Programming languages do not have the importance in practise as one might think when reading publications or listening to discussions at conferences. Actually most people working in big projects are familiar with several languages and except for very tricky programming, which is to be avoided anyway, they produce acceptable software. But this is mainly true for programming "in the small". For global design and construction, the language environment and the experience of programmers is much more important than the programming language. Meanwhile the first packages supporting programming "in the large" are reaching the market.

As a matter of fact, years ago in the scientific community many application packages had been written in PASCAL. In the meantime most of the more successful have been rewritten in FORTRAN, in order to exploit better language environment and execution speed as well as portability.

Databases

There are different requirements concerning databases and their management systems at the different levels of a control system. Therefore it seems quite natural to apply different database structures and access methods at each level. At the lowest level the databases consist of tables which are downline loaded by higher level elements and which are updated in realtime. Part of the table contents are continuously updated or read via the communication system.

At the process control computer level, realtime database systems are required. Fast access and proper data interfaces to lower and higher level databases are required. The structures are relatively stable, which means that read and write accesses by far exceed insertions, deletions or additions; therefore access algorithms are of high priority and can be adapted to the table structures. The table structure generation can be a kind of compilation. The amount of data often does allow memory resident solutions.

For the high level database, three features are required: Firstly, the volume of data is increasing (to more than gigabytes). Secondly, the structures must be very flexible. It must be possible to add attributes or change dependencies whilst the database is in operation. Thirdly, the database must be accessible by many users or programs at the same time. Such requirements are fullfilled by relational databases. Their only disadvantage is low speed. For this reason a careful evaluation is needed.

A proper cooperation between the databases mentioned may lead to a satisfying solution where structures are defined and changed at the highest level which also hold static data and where elements of the lower levels can be compiled and down line loaded. Saving of low level element tables into the relational database can be done triggered by events or time marks where slow responses are not important (Fig. 11).

Integration of software

One of the most challenging tasks for the future of large systems is integration. There are several reasons why integration is required and why it is taking place now or will be necessary in the near future. The fast progress of control technology is forcing companies to think about stable interface ports in hardware and software and secondly large systems very often enforce vendor mixing where standards with wide acceptance are a must.

As an example of standardisation efforts the MAP project initiated by General Motors must be mentioned. Nowadays this company owns about 50,000 factory floor programmable systems and this number will grow to more than 200,000 by end of this decade. There are about 15 large vendors in creating and testing a communication standard between all factory floor systems and minicomputer systems of the higher level.

Besides communications, languages, database systems and tools are being standardised on a world wide scale.

Standards and their misuse

Having the huge task of integration in mind, everybody is in favour of standards which actually have been successful at all levels and worldwide. But there is also some danger that standards are misused. Let me explain this by an example. A graphics package is announced to be GKS compatible - what does this mean? Which level of GKS? Eventually we find out that calls of the package are transformed into GKS calls therefore being able to drive GKS compatible peripherals - but no program issuing GKS calls could use this package.

The complexity of software standards requires additional information like precise specification of test and validation programs especially addressing performance measures like hardware requirements and execution times.

Education and training of software producers

The desired kind of education and the training necessary highly depends on the project phase and the level within the control hierarchy. In our environment we find two types of team members.

Firstly, young people coming from schools. Here the main problem is that universities or engineering schools in general do not teach system design or programming "in the large" nor how to evaluate products or cooperate with vendors.

Secondly experienced old fighters. They have difficulties with new technologies and like their personal style of work which may be different from a common working style based on a toolset.

My personal opinion is that conventional courses are not the right way of education and training; instead I think that small projects or well defined subprojects at an early stage of a project are the right way. Practical work has priority over theoretical courses.

Manpower and costs

Instead of a detailed analysis of these items which is not feasible in the time available let me give you some figures gathered by experience and some hints which we regard as useful.

For the low level control systems, the close cooperation of programmers and system engineers is essential. It is efficient to put these programmers into the groups working on the subsystems and let them support all testing of the components to be controlled.

For the higher levels, teamwork is a must, using the same tools and referring to the same definitions and formats. Complex software for microcomputer systems should be written as cross software.

For the highest levels of the control software, the cooperation of application experts in the software team is desirable. These people are also well suited to be later the supervisors for an operator team.

Evaluation of software products takes at least 6 man months and may go up to one or two man years if it is done properly. Probably the results of such evaluations will find a wider distribution in the future. An interesting form of early evaluations are field tests, because for such tests the software is free of charge.

The cost of system software products like compilers, utilities, database systems etc. is still relatively low compared to tools and application packages. The former are in the order of 20-50 KDM while the latter go up to several hundred KDM.

I am not addressing the problem of hiring or leasing programmers, but although this is expensive and it is difficult to integrate these team members, there is no other solution provided that the market situation does not change. It is extremely difficult to get experienced programmers.

The production of software is going to be very expensive so that the emphasis must be laid on purchasing whatever fits the requirements and concentrating on interfacing and integration.

CONCLUSION

Software has become a very important part of a project and the resulting technical system. Its managment is a complex undertaking from the very beginning of the project far into the maintenance and development phase. Project leaders are well advised to have close contacts with software technology in order to benefit from its services and avoid the chaos caused by inefficient organisation and integration.

LITERATURE

1) Experience from the Implementation of a Control- and Data Acquisition System for the Tokamak TEXTOR
J.A. Boks, H. Halling
Proc. of the 9th Symposium on Fusion Technology, Chicago, Oct. 1981

2) Database Management in a Distributed Process Control System
J.R. Crichton, M.G. Willey
Proc. IFAC Real Time Programming 1983, Hatfield, UK

3) The LEP Control System
LEP Design Report, CERN-LEP/84-01, June 1984

4) Database Requirements of LEP Instrumentation
C. Hovet et al.
LEP Note No. 467 CERN

5) Controlling an Accelerator - The Operations Viewpoint
V. Hatton, G. Shering
CERN/SPS 80-12 (Internal Report)

6) Measurements on the Control Network of the SPS
J. Altaber, J.P. Jeanneret
CERN/SPS/ACC 78-1

7) Two Years of Experience with the PETRA Control System
DESY M-81/08 Internal Report

8) Nova Laser Alignment Control System
Paul J. van Arsdall et al.
UCRL-90033, Lawrence Livermore NL

9) Supervisory Control and Diagnostics System for the Mirror Fusion Test Facility
P.R. McGoldrick
UCRL-53090, Lawrence Livermore NL

10) The Nova Control System: Goals, Architecture, and System Design
G.J. Suski et al.
UCRL-86827, Lawrence Livermore NL

11) Strategies of Design, Development and Activation of the Nova Control System
F.W. Holloway
UCRL-89552, Proc. of the 10th Symposium on Fusion Engineering, Philadelphia 5-9, 1983

12) SNQ Projektplan
Technische Beschreibungen der Anlagenteile
Leittechnik, Dez. 1984

KFA Internal Reports for Layout of a System and Evaluation of Software

13) SNQ-Controls Overview
H. Halling
Sept. 1983

14) Software für SNQ-Kontrollsystem, Organisation und Strukturierung
H. Halling, K.D. Müller, H. Stoff, W. Tenten, K. Zwoll,
K. Pütz (Ingenieurbüro Pütz), D. Seltge (Bionatik, Freiburg)
Dez. 1983

15) Measured Transmission Performance of an Ethernet Link
W. Erven, S. Trencseni, K. Zwoll
Mai 1983

16) Merkmale von Datenbank-Management-Systemen, Einsatzmöglichkeiten in technisch-orientierten Projekten sowie Zeitmessungen an einem relationalen Datenbank-Management-System
K. Pütz
Okt. 1983

17) VAX Lan, Down-Line Loading and Remote Debugging über ETHERNET
H. Stoff, H. Heer
Juli 1984

18) Concurrent CP/M86 für Microprozessoren INTEL 8086
Übersicht über das Betriebssystem. Realtime Messungen zwischen Prozessen
W. Erven, W. Gruhn
Okt. 1984

19) Überlegungen und Untersuchungen zum Aufbau einer Datenbankanwendung für das SNQ-Rechnerkontrollsystem mit Hilfe des relationalen Datenbank-Management-Systems VAX-Rdb/VMS
M. *Rudolph, K. Pütz*
März 1985

20) VAX LN Pascal - An Outline of its Non-Standard Features
K. Andrews
März 1985

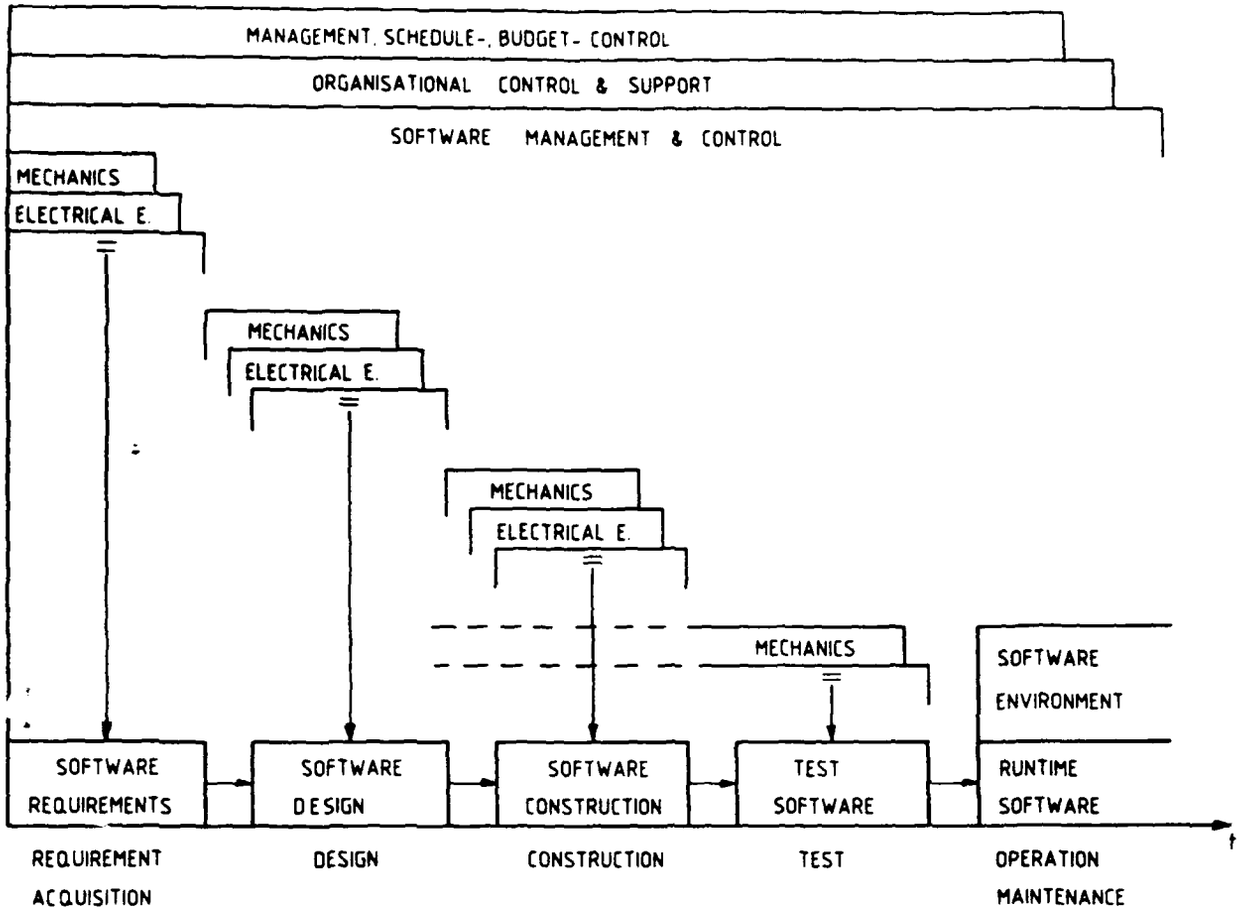


FIG. 1 PROJECT PHASES AND SOFTWARE

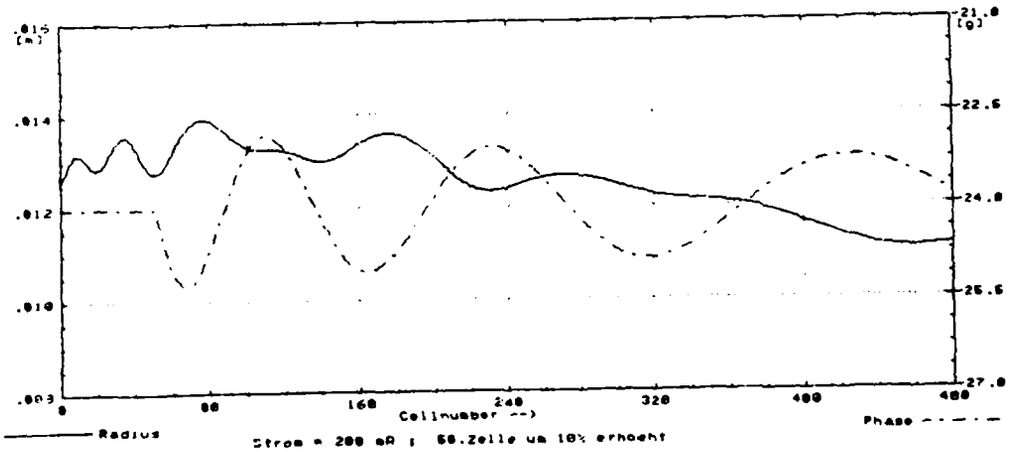


FIG. 2 SIMULATION

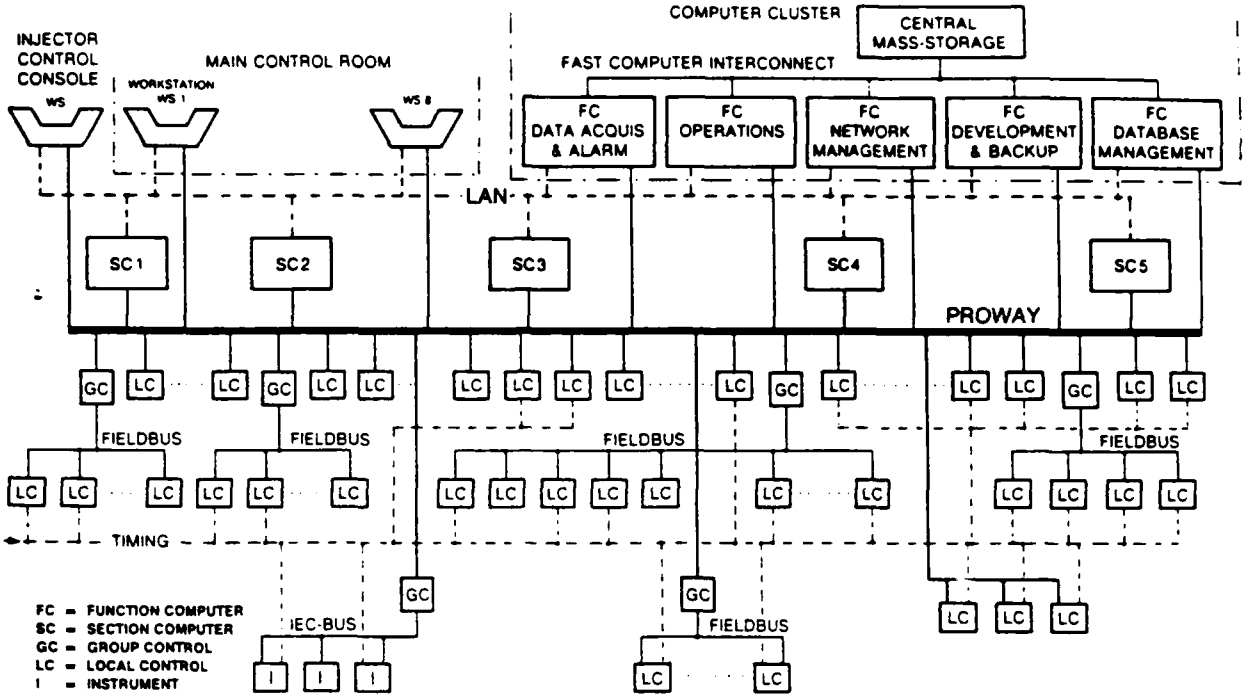


Fig.3 STRUCTURE OF THE COMPUTER CONTROL SYSTEM

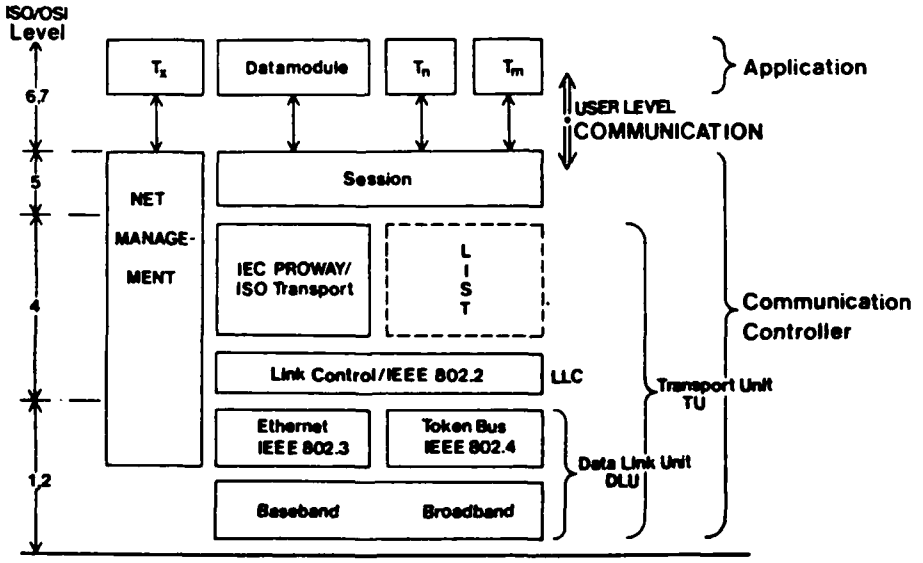


FIG.4 STANDARDS FOR COMMUNICATION

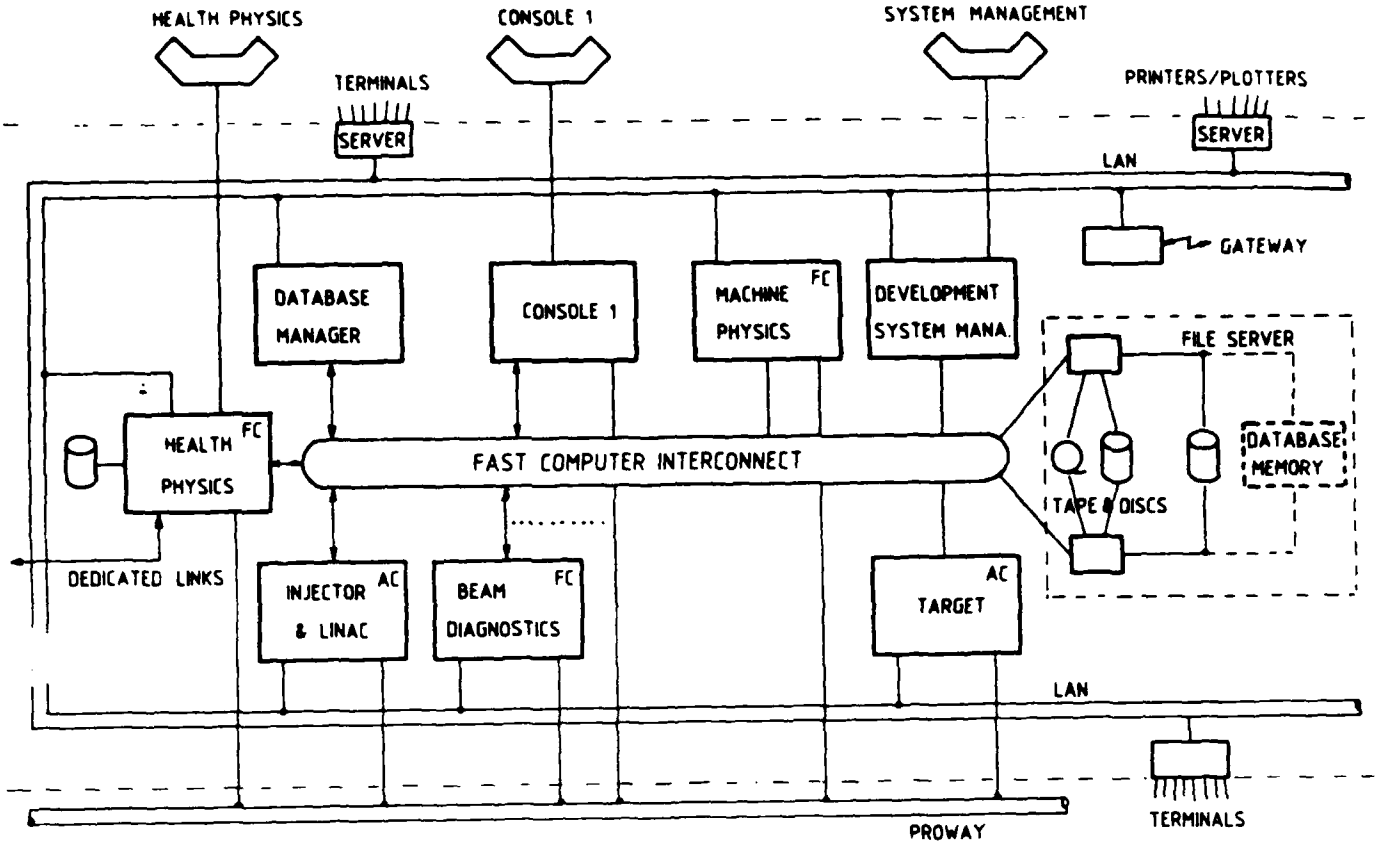


FIG. 5 CENTRAL COMPUTER COMPOUND

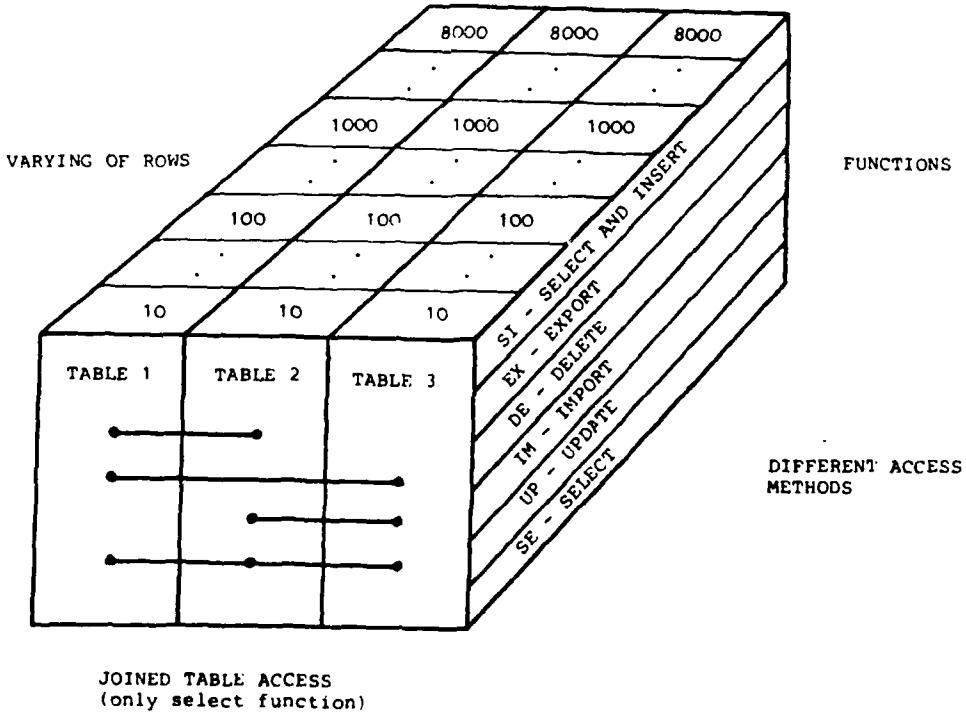


FIG. 6 STRUCTURE OF TESTABLES

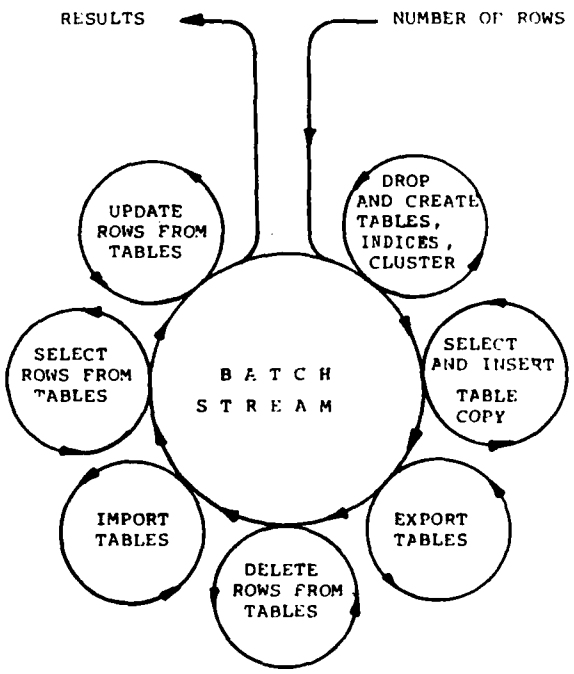
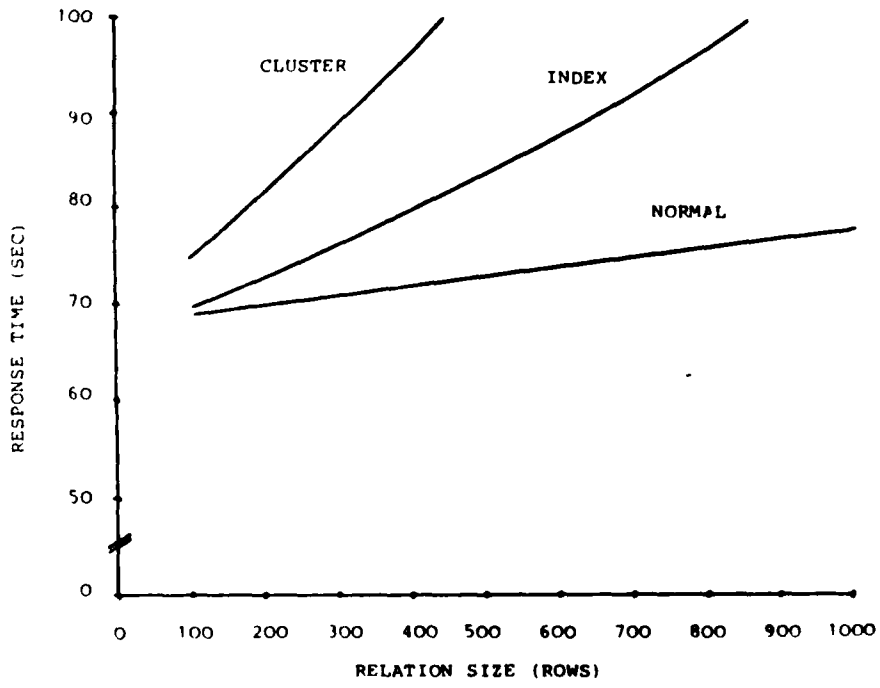


FIG. 7 TEST SEQUENCES



SELECT ALL ROWS FROM SINGLE TABLE) COPY DATA
AND) FROM ONE TABLE
INSERT VALUES FROM ALL ROWS INTO) TO ANOTHER
NORMAL-/INDEX-/CLUSTER-STRUCTURED TABLE) TABLE

FIG. 8 RESULTS

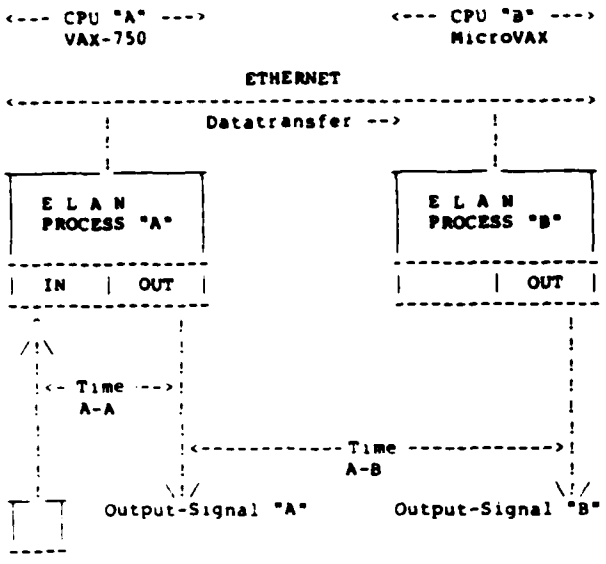


FIG 9 TEST SETUP

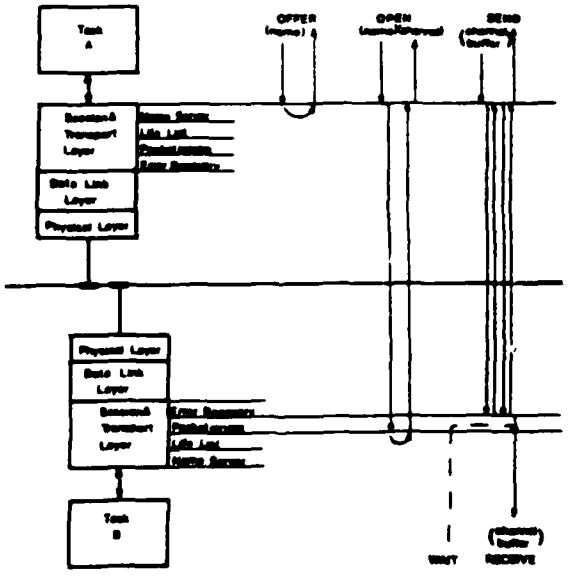


FIG. 10 TEST DESCRIPTION

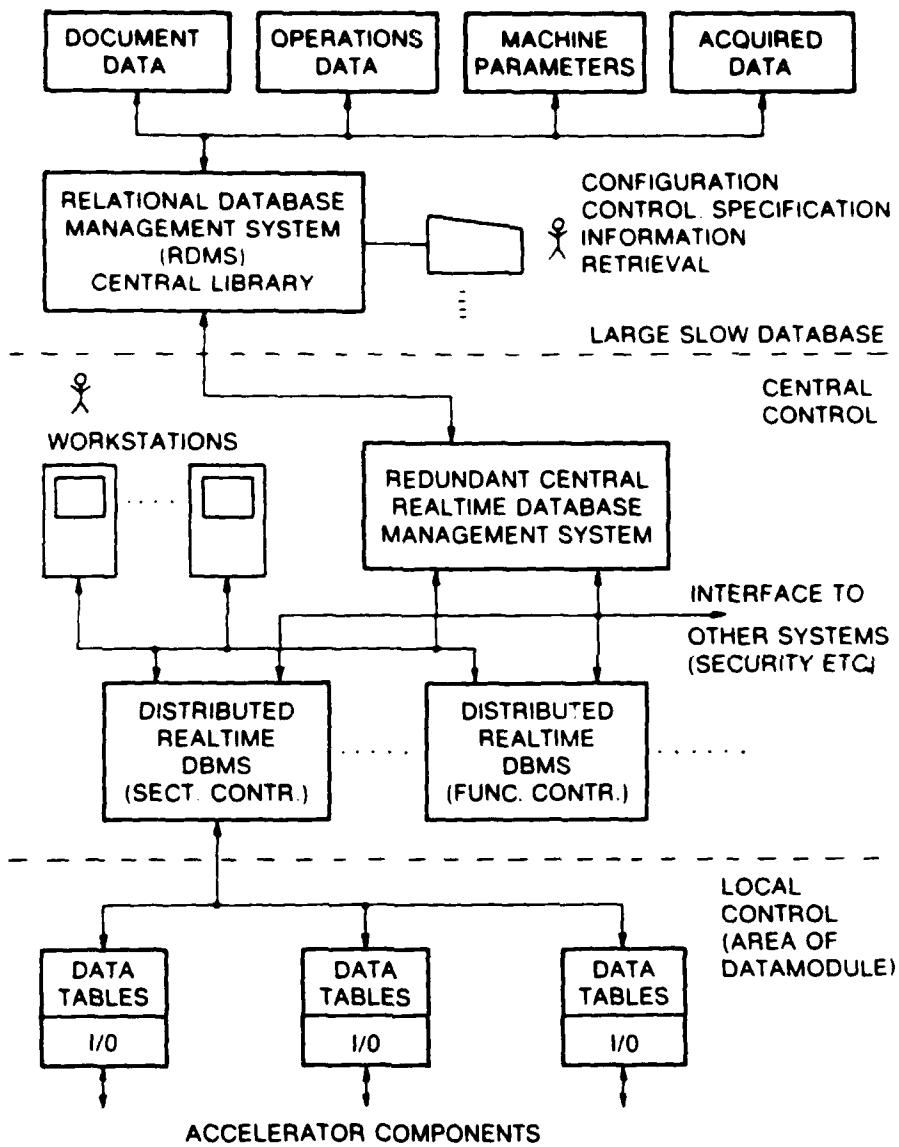
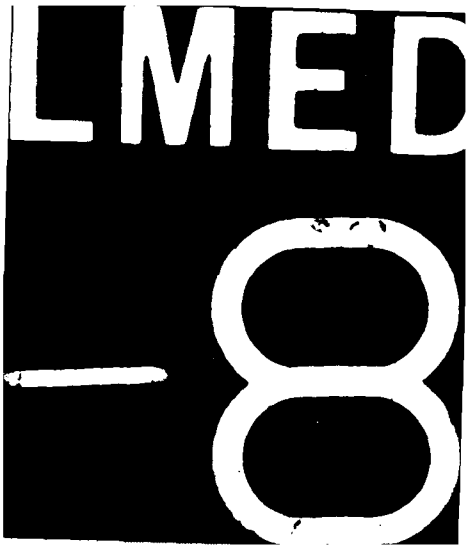


Fig.11 DATA ORGANISATION


LMED
8